ON REDUNDANCY AND UNTESTABILITY IN SEQUENTIAL CIRCUITS

BY

MAHESH ANANTHARAMAN IYER

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Electrical Engineering
in the Graduate School of the
Illinois Institute of Technology

Approved ______________________________
Thesis Advisor

____
Academic Advisor

Chicago, Illinois
July, 1995

To

The Memory of my Dear

AMMA

Smt. GIRIJA ANANTHARAMAN

ACKNOWLEDGMENT

I would like to take this opportunity to thank a few people without whose participation this work would not have been possible. Firstly, I would like to express my most sincere gratitude to Miron for having given me the opportunity to work with him, for being an excellent guide, for always being concerned about my welfare at IIT, for having taught me over the years how to handle tricky situations that are often encountered, for considering me part of his lovely family, for letting me land on his dinner table on several occasions, and for being a very understanding friend. My discussions with Dr. David Long proved to be excellent and highly fruitful. It let me understand my Ph.D. problem better and come up with reasonable solutions in reasonable amounts of time. I really enjoyed working with David and would like to sincerely thank him for having given me that opportunity. I would also like to thank Dr. Henry Stark for having given me the opportunity to pursue graduate studies at IIT. I also want to thank my academic advisor, Prof. John A. Nestor, for his active support and encouragement during the entire course of my graduate work at IIT and for doing an excellent job in organizing my thesis committee.

My special thanks to Prof. Irith Pomeranz and Prof. Sudhakar Reddy from the University of Iowa for many many useful e-mail arguments that helped finding an error in the initial theory of the ideas presented in this thesis. Also, thanks to Prof. Tim Cheng, Dave Miller, Srimat Chakradhar, Vishwani Agrawal, Fadi Maamari, Scott Davidson, and Shianling Wu for many useful discussions. Specially, I would like to thank Chak and Vishwani for being constant sources of encouragement and for having given me the opportunity to work with them over two summers in New Jersey. I would also like to thank a few other people; Jim Coplein for discussions on C++ and Anoop Singhal for helping me with the DDB package of AT&T. I also thank AT&T Bell

Laboratories and the ECE department of IIT for supporting this work. The support provided by Prof. Peter Clarkson, Scott Davidson, Alfred Dunlop, Jean Dussault, Hao Nham, Prof. Donald Ucci, and Shianling Wu made this work possible and is gratefully acknowledged. The support provided by Al Dunlop during the Fall of 1994 gave me the opportunity to interact with David Long and was very crucial in the final stages of this thesis.

The inspiration offered by my Amma will always be cherished. I want to specially thank my Appa, Venkat, Rajee, and Patti for their patience and understanding during the entire course of this project. I love you all.

I thank all my friends at IIT for making my stay here a memorable one. Lastly, I want to thank my better half, Shripriya, for her patience, understanding, and love without which I could not have completed this thesis.

M. A. I.

TABLE OF CONTENTS

APPENDIX Page

LIST OF FIGURES

Figure Page

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation	Term
ATG	automatic test generation
ATPG	automatic test pattern generation
BDD	binary decision diagram
c-v	controlling value
DFT	design for testability
FF	flip-flop
FILL	algorithm to find illegal states
FIRE	fault-independent redundancy identification algorithm
FIRES	fault-independent redundancy identification algorithm for sequential circuits
FOB	fanout branch
FMI	fault that modifies initialization
FPI	fault that prevents initialization
FUNI	algorithm to find untestable faults using illegal states
FUNTEST	algorithm to find untestable faults
I/O	input/output
MTBF	mean time between failures
PO	primary output
PI	primary input
RID	redundancy identification
RTG	random test generation
SCC	strongly connected component
SFT	single-fault theorem
SSF	single stuck-at fault
STG	state transition diagram

Abbreviation	Term
TG	test generation
VLSI	very large scale integration

ABSTRACT

This research addresses the problems of identifying redundancy and untestability in synchronous sequential circuits. These problems are important components of today's complex design flows, both in synthesis and in automatic test generation (ATG). They are NP-complete problems and most of the work done so far provide solutions based on ATG. The widespread misconceptions about redundancy are revealed and examples are provided to illustrate the differences between untestability and redundancy in sequential circuits. Existing techniques to identify sequential redundancy are discussed, and it is shown that some of these methods are based on incorrect theoretical results. Specifically, it is shown that untestable faults in balanced pipeline circuits and in circuits with fault-free global reset mechanisms are not necessarily redundant, and that a constant function (i.e. a signal that is always 0 or 1 after initialization) does not always indicate a redundancy. It is also shown that adding a global reset mechanism or retiming synchronous circuitry may introduce redundancies. In addition, the existing solutions rely on unrealistic simplifying assumptions, or are applicable only to small circuits, or identify only special classes of faults. A common limiting factor is that they are based on exhaustive search done by ATG algorithms, which results in very large computational requirements.

This research presents the first efficient algorithm (FIRES) to identify sequential redundancy, that is practical for large circuits. No global reset is assumed and no state transition information is required. The concept of a *c*-cycle redundancy as a generalization of the conventional notion of sequential redundancy is introduced, and it is shown that faults identified by FIRES are *c*-cycle redundant. FIRES is based on the important result that a fault which requires a conflict as a necessary condition for its detection is *c*-cycle redundant. Removing *c*-cycle redundancies allows circuit simplifications which are not possible with conventional redundancy. FIRES has

provably polynomial-time complexity and is practical for large circuits. However, it is not guaranteed to identify all redundancies in a circuit. Experimental results on benchmark circuits indicate that FIRES finds a large number of redundancies. When a sequential test generator targets a sequentially redundant fault, the most it can do is prove it as untestable, but it cannot distinguish between untestability and redundancy. We show that, in general, the redundant faults identified by FIRES are not easy targets for state-of-the-art sequential test generators.

Another algorithm, FUNI, to find untestable faults using illegal states is also introduced. FUNI uses the illegal states identified by a recently proposed algorithm, FILL, and finds untestable faults for which these illegal states are necessary for detection. FUNI is a fault-independent and direct method that identifies untestable faults without using the exhaustive search involved in ATG. Experimental results show that FUNI finds a large number of untestable faults up to several orders of magnitude faster than an ATG algorithm that targeted the faults identified by FUNI. Also, many untestable faults identified by FUNI were aborted by the test generator.

CHAPTER I

INTRODUCTION

1.1 Automatic Test Generation

With the advances in VLSI technology, the field of *testing* has achieved much importance. Testing refers to the verification of the functionality of the system under consideration. An unexpected response at the output(s) of the system reflects a misbehavior in the functionality of the system. A second goal would be to *diagnose* the system to locate the cause of misbehavior. These experiments of testing and diagnosis are necessary to ensure that the manufactured product meets all its functional and timing requirements and to increase the *mean time between failures (MTBF)* factor of the system. However, testing a system is also regarded as the bottleneck in the manufacturing process of the system, due to its inability to keep up with the rate of manufacture of the system and is hence regarded as a necessary area of research.

Test Generation (TG) is the process of determining the stimuli, known as *test vectors*, necessary to test a digital system. This digital system may refer to combinational or sequential circuits, though the methods for generating tests for both of these classes of circuits will be different. This is because in combinational circuits, the output logic values depend only on its present input values, whereas in a sequential circuit the present and past values will have to be processed to determine its current output. *Automatic Test Generation* (ATG) refers to TG algorithms, that given a model of a system, can generate tests for it. TG can be *fault oriented* or *function oriented*. In fault-oriented TG, one tries to generate tests that will detect (and possibly locate) specific faults. In function oriented TG, one tries to generate a test that, if it passes, shows that the system performs its specified function. A normal requirement of these tests is that they detect a very high fraction of the modeled faults. The detected fraction of faults is called the *fault coverage*. Also, since the same set of test vectors will be applied to a large number of copies of the chip, short test sequences are desirable.

A digital system consists of many printed wiring boards, each containing several VLSI chips. Testing such systems can be an overwhelming task and faster and more efficient test generation algorithms are needed. The variety of subtle failures that can occur in a VLSI chip further complicates the testing process. Complexity of test generation is enormous.

1.2 Errors and Faults

An unexpected response of the system being tested is referred to as an *error*. These errors may be caused due to *design errors, fabrication errors, fabrication defects,* and *physical failures*. Fabrication errors, fabrication defects, and physical failures are collectively referred to as *physical faults*. These physical faults may be classified as *permanent, intermittent or transient*, based on their stability in time. To aid a thorough mathematical treatment of testing and diagnosis, these physical faults are conveniently represented by *logical* faults. Based on TG, these faults are often classified as *testable (or detectable) and untestable (or undetectable)*. A fault is detected by observing an error caused by it. A fault is untestable if an error caused by it cannot be observed. The basic assumptions regarding the nature of logical faults are referred to as a *fault model*. The most widely used fault model is the single-stuck-at-fault (SSF) model, in which it is assumed that a single line in the circuit is permanently stuck at a logical value.

1.3 Testing for Single-Stuck Faults (SSFs)

Faults defined in conjunction with a structural model are referred to as *structural faults* and their effect is to modify the interconnections among the components in the circuit. Thus, structural fault models assume that components are fault-free and only their interconnections are affected. Typically, these faults may be *shorts* or *opens*. A *short* is formed by an unintended interconnection between certain lines in the circuit, whereas an *open* results from the breaking of a connection. The corresponding logical fault consists of the signal being stuck at a fixed value v, $v \in \{0, 1\}$, and is denoted by *s-a-v*. Test generation is a commonly recognized complex problem with many interacting aspects for trade-offs. These aspects mainly include the cost of TG, the quality of the generated test and the cost of applying the test. TG can be mainly classified as *random* or *deterministic*. Random TG (RTG) only involves the generation of random test vectors. Deterministic TG generates tests by processing a model of the circuit. ATG is deterministic TG. Though, RTG is a simple process, it requires a large set of vectors and it may be expensive to measure the quality of the generated test. Deterministic TG, on the other hand, produces shorter and high-quality tests. Deterministic TG can be fault-oriented or fault-independent. In fault-oriented TG, tests are generated for specified faults of a fault universe (defined by an explicit fault model). This is commonly referred to as *targeting* faults and generating tests for them. Fault-independent TG works without targeting individual faults from the fault universe. The generated tests include both the stimuli to be applied and the expected response of the fault-free circuit.

In fault-oriented ATG, two basic steps are involved for generating a test for a target fault. These include *activating* the fault and *propagating* the resulting error to a primary output (PO). Fault activation refers to exciting the fault (*l s-a-v*) by setting the primary input (PI) values that cause the line *l* to have a value $\overline{v}$. The process of determining the PI values for the target fault is referred to as a *line-justification* problem. Typically, a *fault-effect* is said to have been created and is denoted as v/v_f. These composite logic values represent the values of the line *l* in the fault-free and faulty circuits. Fault propagation involves the propagation of this fault-effect to a PO, thus making this fault observable. If this process succeeds, the fault is referred to as detected and the test that detects this fault is the resulting PI values set up during fault activation and propagation. On the other hand, if the process fails, the target fault is referred to as an untestable fault, and in this case the outputs in the fault-free and faulty circuits are the same. Though untestable faults may appear harmless, a circuit with an untestable fault may invalidate the single fault assumption. This is because the single fault assumption assumes a *frequent testing strategy*, in which we detect a fault before a second one occurs and this may not be possible if an untestable fault is encountered. The goal of ATG is to generate a *complete detection test set*, which is a set of tests that detect any detectable fault. However, Friedman [Friedman 1967] has shown that a complete detection test set may not be sufficient to detect all detectable faults if an untestable one is present in the circuit.

1.3.1 Combinational ATG. The test generation problem for combinational circuits belongs to the class of NP-complete problems [Ibarra and Sahni 1975]. Several algorithms have been proposed in the literature for deterministic combinational ATG. Most of these algorithms are fault-oriented and are branch-and-bound type of algorithms. If given enough time, they are guaranteed to find a test for a fault, if one exists. They use a decision tree mechanism and make decisions about values of signals in the circuit. In case a failure occurs, by encountering a conflict, backtracking is enforced to reverse the most recent decision. The process continues until a test is found or the entire search space is exhausted or the number of backtracks exceeds the prespecified backtrack limit. Several heuristics have also been proposed to speed-up combinational ATG algorithms. These heuristics mostly comprise of static and dynamic learning techniques that work as preprocessing or during ATG. They attempt to reduce the search space during ATG.

Combinational ATG is applicable only when dealing with combinational circuits or with sequential circuits that are designed using a full-scan design methodology. Full-scan circuitry typically adds a significant area overhead to the circuit. It also imposes a performance penalty. An additional drawback is that scan vectors need to be scanned in and the results scanned out; thus the circuit cannot be tested at speed.

1.3.2 Sequential ATG. ATG for sequential circuits also belongs to the class of NP-complete problems. However, sequential ATG is much more complex than combinational ATG. The combinational ATG problem is only a subset of the sequential ATG problem. Most sequential circuit ATG algorithms use modified versions of combinational ATG algorithms by using the combinational iterative logic array model of the sequential circuit. In the rest of this thesis, we restrict ourselves to synchronous sequential circuits whose components are gates and clocked flip-flops (FFs).

Two test strategies exist for deterministic sequential ATG. The first, called the *single observation time* strategy, relies on finding a test sequence for a fault such that the fault-free response is different from the faulty response at some specific vector i on a specific PO, independent of the initial states of the fault-free and the faulty circuits. The second, called the *multiple observation time* strategy [Pomeranz and Reddy 1992], is based on the concept of strong detection [Abramovici *et al.* 1990], which relies on finding a sequence for a fault by making the fault-free circuit starting at state S_1 and the faulty circuit starting at state S_2 exhibit different output responses at some vector (not necessarily the same), for all (S_1, S_2) pairs. However, for large circuits this strategy is computationally expensive. Some faults that have no tests under the single observation time strategy may have tests under the multiple observation time strategy. In practice, most sequential ATG systems are based on the single observation time test strategy.

Sequential ATG is complex because the FFs in the circuit are not directly controllable and observable. A test for a fault must be such that it works for all initial (power-up) states of the circuit; i.e. the test cannot rely on particular initial states of the circuit. Several techniques have been proposed to solve the sequential ATG problem. These techniques use combinations of forward and reverse time processing for line justification, fault propagation and state justification. Some techniques also build state transition diagrams (STGs) and some others make the assumption about the existence of a reset state. Nevertheless, practical sequential ATG programs tend to have huge memory and time requirements and often produce results that are far from those desired.

1.4 Fault Simulation

Fault simulation plays an important role in grading a test, often referred to as *test evaluation*. This process measures the fault coverage for a given test, which in turn determines the quality of the test. Fault simulation computes the response of the circuit in the presence of faults to the test being evaluated. Generally speaking, fault simulation techniques are computationally less expensive than fault-oriented TG. Thus when using fault-oriented TG, fault simulation techniques are very useful to determine other faults detected by the test generated for the current target fault. This helps in reducing the number of target faults that the test generator has to deal with.

1.5 Why Do We Have Redundancy ?

The design of any logic circuit may be done manually or with the aid of logic synthesis tools. In either case, some unintentional logical redundancies may be introduced in the circuit. This is because any digital circuit may have many different implementations realizing the same function. Global redundancies can also be unintentionally introduced (without the knowledge of the designer) when merging individually optimized blocks of logic.

Sometimes, redundancies are purposely included in the design to realize a fault tolerant system. However, these systems are almost untestable for off-line testing. The solution is to remove the redundancy for off-line testing. Redundancies may also be introduced in a circuit to avoid hazards. This design technique is illustrated in Figure 1.

The function implemented by the circuit shown is $z = ab + bc + \bar{a}c = ab + \bar{a}c$. Thus the gate Y that produces the term bc is not logically required. However, removing this gate produces a static hazard, as a spurious 0-pulse may appear at the output when the input vector changes from 111 to 011. Gate Y ensures that the output remains constant at a value 1, during this transition. However, the fault Y *s-a*-0 is untestable.

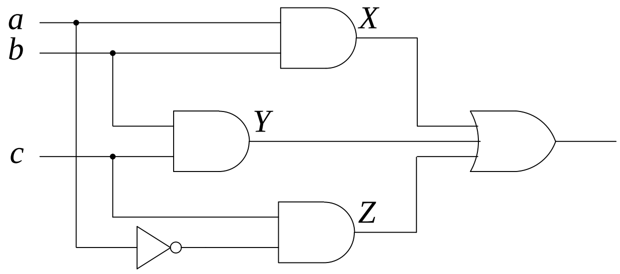


Figure 1 . A Hazard-free Circuit

1.6 Why Worry About Redundancy ?

Redundancies in logic circuits complicate ATG and increase the chip area and often the propagation delays in the circuit. A redundant circuit can always be simplified by removing at least one gate or gate input. Moreover, redundancy identification (RID) is important if the desired fault coverage is to be achieved.

Another detrimental effect of redundancy is that it may unnecessarily reduce the yield of the IC manufacturing process [Davidson 1994]. For example, many sequentially redundant faults become detectable with full-scan testing, and many sequentially and combinationally redundant faults become detectable with I_{DDQ} testing. Although the circuit remains fully operational in the presence of a redundant fault, full-scan testing or I_{DDQ} testing will reject that faulty circuit and result in a yield loss.

1.7 Fault Detection and Redundancy

The RID problem and the fault detection problem are closely related. An untestable fault in a combinational circuit implies that there exists no test for the fault, such that a boolean difference can be observed at the POs of the circuit. The presence of such a fault does not affect the input/output (I/O) behavior of the circuit. Thus, a combinational circuit that contains an untestable fault is said to be redundant, since such a circuit can always be simplified by removing at least one gate or gate input. Typically, this fault is referred to as a *redundant fault*. The general concept of redundancy is broader than the particular one related to the existence of untestable stuck faults, and it denotes a circuit that can be simplified. A general type of redundancy [Hayes 1976] exists in a circuit when it is possible to cut a set of r lines and to connect $q \leq r$ of the cut lines to some other signals in the circuit without changing its function. This also includes other types of redundancies, not related to untestable stuck faults, like a string of inverters. Replacing these string of inverters by a single line (with or without inversion as the case may be) also yields the same function.

The presence of a redundant fault may invalidate a complete detection test set. Other problems that can arise in redundant circuits include the following [Friedman 1967]:

— If f is a detectable SSF and g is an untestable SSF, then f may become untestable in the presence of g. Such a fault f is called a *second-generation redundant fault*.

— Two untestable single faults f and g may become detectable if simultaneously present in the circuit. In other words, the multiple fault $\{f, g\}$ may be detectable even if its single-fault components are not.

Any deterministic TG algorithm can identify all the single redundant faults in a combinational circuit, if given enough time. However, as is commonly recognized, TG for combinational circuits is an NP-complete problem [Ibarra and Sahni 1975]. The search process of TG involves decision processes at various levels. An inconsistent decision, in its pursuit to find a solution, leads to a *contradiction* or *conflict* which is solved by initiating a *backtracking strategy*. This strategy ensures the exploration of the complete space of possible solutions and recovery from incorrect decisions. The complexity of computations during TG is measured with respect to the number of backtracks involved and the test generator exhibits its worst-case behavior in the case of redundant (untestable) faults which do not have any tests. Thus redundancy identification (RID) using ATG is *indirect* and can be viewed as a byproduct of TG.

In contrast, a *direct* RID technique finds redundancies without the search process involved during ATG. Direct RID techniques can be classified as *static* or *dynamic*. Static RID techniques work as preprocessing to ATG, whereas dynamic RID techniques work during ATG. The advantage of these low-cost techniques is that redundant faults identified by using them can be excluded from the universe of faults the test generator has to deal with, thus significantly reducing the CPU time used in TG. Also, redundancies identified using low-cost techniques can be useful in synthesis for circuit simplification.

Unlike combinational circuits, an untestable fault in a sequential circuit is not necessarily redundant [Abramovici and Breuer 1979]. Typically, such faults affect the initialization of the circuit. In a sequential circuit, redundant faults form a subset of the untestable faults. Identifying sequential redundancy using ATG has the added complexity of verifying whether an untestable fault is indeed redundant. In general, a practical sequential circuit test generator cannot distinguish between untestable and redundant faults. It relies on finding a test for a target fault, independent of the initial state of the circuit. When it fails to find one, after exhausting the search-space, the fault is considered untestable. Thus, as stated earlier, the test generator exhibits its worst-case behavior in the case of untestable faults. Moreover, a practical sequential test generator cannot even be trusted to reliably identify untestable faults. It is well known that because of the pessimism involved in the use of the unknown logic value, a test generator may report as untestable, a fault for which a test sequence in fact exists

[Abramovici *et al.* 1990]. While in ATG this pessimism may only lead to fewer faults being tested, it has serious implications when identifying redundancy, because removing the region associated with an irredundant fault results in a circuit whose behavior is different from that of the original circuit. Nevertheless, most of the previous work on sequential RID and on identifying sequentially untestable faults [Cheng 1991a, Cheng 1991b, Cheng 1993, Cho *et al.* 1993c, Agrawal and Chakradhar 1993, Entrena and Cheng 1993, Pomeranz and Reddy 1994a, Pomeranz and Reddy 1994b, Dey and Chakradhar 1994, Liang *et al.* 1994] is based on ATG and hence require large computations. Moreover, some of these solutions unrealistically assume the existence of a fault-free global reset mechanism. In general, the sequential RID problem is considered very complex and has no practical solution.

Once a redundant fault is identified, the circuit can be simplified by removing the redundant structures associated with that fault. The procedure is to insert the fault in the circuit and remove all the logic that no longer influences the outputs. This relies on the fact that the function of a circuit does not change in the presence of a redundant fault. When the region for a redundant fault is removed from the circuit, the function of the circuit is not altered but other redundant faults may become irredundant and some irredundant faults may now become redundant. However, if the region for every redundant fault is removed as soon as the fault is identified, it is not possible to make a previously identified redundant fault irredundant, as it is no longer present in the circuit. The problem that remains to be solved is the identification of newly introduced redundancies, as a result of the current removal. Thus the problem of redundancy removal is considered more complex than the RID problem itself.

1.8 Overview of This Thesis

This thesis presents novel and practical algorithms for combinational and sequential RID and for identifying sequentially untestable faults. These algorithms do not rely on test generation search effort and do not assume the existence of a fault-free global reset mechanism. These techniques are useful for both logic synthesis and test generation, as they will be useful to produce a more area-optimized and a more testable version of the original circuit. Moreover, non-ATG-based techniques to identify redundancy and untestability can improve the performance of the test generator by reducing the universe of untestable faults it has to deal with. More importantly, the faults found by these algorithms are untestable under the multiple observation time test strategy. With respect to test generation, this has significant advantages because these faults will remain untestable no matter how sophisticated the test generator is. Design for testability (DFT) techniques attempt either to make the test generation simpler or to decrease the number of untestable faults. Several DFT techniques exist in the literature. The algorithms presented in this thesis could thus serve as good starting points for DFT techniques that aim at increasing the number of testable faults in the circuit with minimal area and performance overheads.

The concept of a c-cycle redundant fault is introduced. A fault is considered as redundant if the behavior of the faulty circuit is indistinguishable from that of fault-free circuit after applying c arbitrary input vectors to the faulty circuit. c-cycle redundancy is a generalization of the conventional notion of sequential redundancy which is represented by the particular case $c = 0$. A new fault-independent algorithm to identify redundancies in sequential circuits without search is introduced. Though the algorithm will be presented with an implicit single clock assumption, it has been efficiently implemented to handle multiple-clock circuits. It will also be shown that this algorithm is the *first efficient solution to the sequential RID problem that is practical for large circuits*. This thesis also presents an efficient algorithm to identify untestable faults using illegal states of the fault-free circuit. All these algorithms identify large numbers of untestable and redundant faults up to several orders of magnitude faster than a practical ATG-based approach to identify untestable faults.

The rest of this thesis is organized as follows. Chapter 2 provides definitions of terminology used in this thesis and presents some classes of sequentially untestable but irredundant faults. Chapter 3 provides an in-depth survey of the literature and describes previously developed techniques. Examples are illustrated to show that some of the existing solutions for sequential RID are in fact based on incorrect theoretical results. Chapter 4 presents a new approach to identify redundancies in combinational circuits without any search and presents experimental results. Chapter 5 introduces the concept of *c*-cycle redundancy and extends the algorithm in Chapter 4 to identify *c*-cycle sequential redundancies. The relation of *c*-cycle redundancy with other existing solutions for sequential RID is also examined and experimental results are presented. Chapter 6 reviews a recently proposed algorithm to identify illegal states in a sequential circuit without assuming a global reset mechanism and presents an efficient algorithm to identify sequentially untestable faults using these illegal states. Chapter 7 shows the relationship between faults found by various algorithms presented in this thesis and presents experimental results to illustrate the combined strength of all the algorithms. Chapter 8 is devoted to conclusions to summarize the contributions of this research and provides directions for future work.

CHAPTER II

BASIC CONCEPTS

This chapter reviews the definitions of some of the related terminology used in this thesis. Combinational and sequential circuits are dealt with in separate sections. The definitions presented are most general and are not limited to any particular logic value system or testing strategy. Examples of some classes of irredundant sequentially untestable faults are also presented.

2.1 Common Terminology

- *Definition 1:* A signal with two or more fanout branches (FOBs) is called a *fanout stem* or simply a stem.
- *Definition 2:* A gate *G* is called a *reconvergent gate* of a stem *s*, it there exist two or more disjoint paths from *s* to *G*.
- *Definition 3:* An input of a reconvergent gate *G*, reachable from its stem *s* is called a *reconvergent input*.
- *Definition 4:* A signal *l* is said to be *bound* by a gate *G*, if all paths from *l* to the POs pass through *G*. *G* is also referred to as a *dominator* of *l*.

Note that there may be more than one reconvergent gate associated with a stem. Fanouts from different stems may also reconverge at the same reconvergent gate. Also, there could be many dominators associated with a line.

2.2 Combinational Circuits

- *Definition 5:* A fault is *testable*, iff there exists an input vector that will produce a different response on some output(s) of the fault-free and the faulty circuits.
- *Definition 6:* A fault is *untestable (redundant)*, iff it is not testable.

Note that in a combinational circuit (or full-scan version of a sequential circuit), all untestable faults are redundant. Once a combinational redundancy is identified, the procedure to determine (and remove) the redundant logic for that fault is conceptually straightforward [Abramovici *et al.* 1990, Abramovici and Iyer 1992, Iyer 1991]. On identifying a redundant fault the *maximum redundant region* for this fault is determined by forward propagating the stuck value of the redundant fault. The entire region bound by the highest-level lines(s) to which the faulty value has been propagated is redundant and can be removed from the circuit. The values propagated to the outputs of this region represent a multiple stuck fault structurally equivalent to the target redundant fault. This procedure ensures that the function of the circuit remains the same after removal of the redundant region and is illustrated in the following example

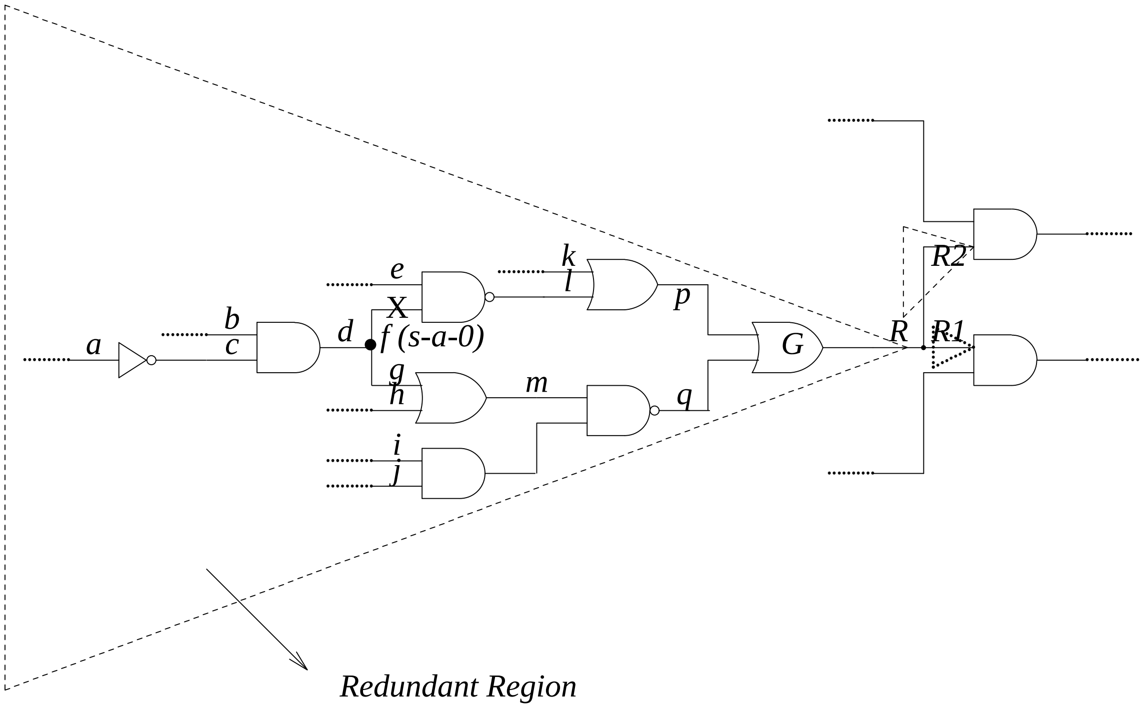


Figure 2. Redundant Region for a Single Redundant Fault

Example 1: Assume that the fault f s-a-0 in Figure 2 is proven to be redundant. This stuck value is then forward propagated as far as possible, up to $R1$ and $R2$. The entire circuit bound by $R1$ and $R2$ can be removed. Note that the multiple stuck fault

{$R1$ *s-a*-1, $R2$ *s-a*-1} is structurally equivalent to the single stuck fault f *s-a*-0. Also, stem d is included in the redundant region as both its fanout branches f and g are included in the region. □

2.3 Sequential Circuits

2.3.1 Legal and Illegal States. The operation of a sequential circuit typically begins by powering up the circuit and applying an initialization sequence. During this initialization phase, the output responses that are not fully defined are usually ignored. After initialization, the circuit has a defined behavior. This defined behavior is what characterizes the normal operation of the circuit. During the normal operation, the circuit always operates in some finite sets of states that satisfy the following conditions.

1. Any state in a set is reachable from any other state in that set.
2. No state in a set can reach a state outside the set.

These finite sets of states are typically referred to as the *terminal strongly connected components (SCCs)* of the circuit. Typically, most sequential circuits are designed to have only one terminal SCC.

- *Definition 7:* A state is *legal* if it is part of any terminal SCC.
- *Definition 8:* A state is *illegal* if it is not legal.

2.3.2 Testable, Untestable, and Redundant Faults. As was stated in Chapter 1, the multiple observation time test strategy is more general than the single observation time test strategy. Faults that are untestable under single observation may have tests under multiple observation. The following definitions from [Pomeranz and Reddy 1993] assume the multiple observation time test strategy.

- *Definition 9:* A fault f is said to be *testable* if there exists an input sequence I such that for every pair of initial states S and S^f of the fault-free and faulty circuit, respectively, the response $Z(I,S)$ of the fault-free circuit to the input sequence I is different from the response $Z^f(I,S^f)$ of the faulty circuit (at some time on some output).

- *Definition 10:* A fault is *untestable* if it is not testable.
- *Definition 11:* A fault is *partially testable* if there exists an initial state S^f of the faulty circuit and an input sequence I such that for every fault-free initial state S, the response of the fault-free circuit to I starting from S, $Z(I,S)$, is different from the response of the faulty circuit starting from S^f, $Z^f(I,S^f)$. (This definition differs slightly from that in [Pomeranz and Reddy 1993], in that it includes testable faults [Iyer *et al.* 1995]. This new definition naturally allows a testable fault to appear as a particular case of a partially testable fault; i.e. a fault is testable if it is partially testable for all initial states of the faulty circuit.)
- *Definition 12:* A fault is *redundant* if it is not partially testable. (A partially testable fault is an irredundant fault.)

The classification of testable, untestable and redundant faults is shown in Figure 3.

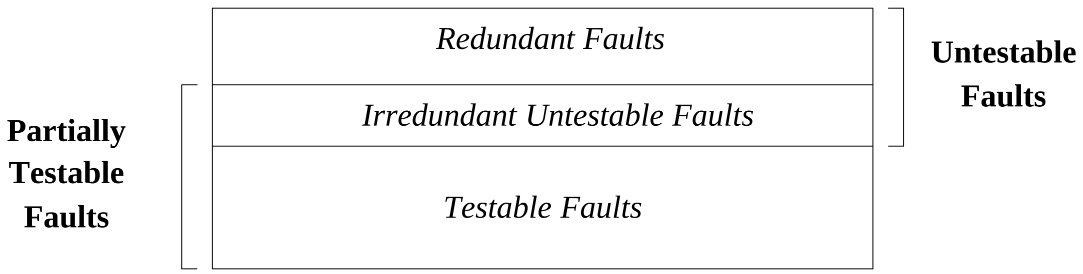


Figure 3. Classification of Faults

A more restrictive definition of redundancy that was used in [Abramovici and Breuer 1979, Moondanos and Abraham 1992] is as follows.

- *Definition 13:* A fault is *redundant* iff the faulty and the fault-free circuits are equivalent, that is, iff every state S^f of the faulty circuit is equivalent to some state S

of the fault-free circuit, and every state S of the fault-free circuit is equivalent to some state S^f of the faulty circuit.

It was shown in [Moondanos and Abraham 1992] that redundant faults under Definition 13 form a subset of redundant faults under Definition 12. More recently, Pomeranz *et al.* [Pomeranz *et al.* 1993] have also shown that there could be redundant faults that do not require the fault-free and the faulty circuits to have equivalent state tables.

2.3.3 Unrestricted and Restricted Test Strategies. The operation of a sequential circuit begins with an initialization sequence. However, using the multiple-observation test strategy, tests can be derived without having to initialize the circuit. Depending on whether the outputs can be observed before or after initialization, tests or partial tests can be derived under different test strategies.

Under the *unrestricted test strategy*, tests or partial tests are derived assuming that the outputs of the circuit can be observed at any time after power-up. Definitions 9-12 are for the unrestricted test strategy. Under the *restricted test strategy*, tests or partial tests are derived assuming that the outputs of the circuit can be observed only after initialization of the fault-free circuit. Accordingly, Definitions 9-12 can be modified. Faults that are redundant under the unrestricted test strategy are also redundant under the restricted test strategy. The converse, however, is not always true as illustrated by the following example.

Example 2: Consider the circuit in Figure 4 and the fault $c1$ *s-a*-1. The fault is untestable because the requirements to detect the fault ($c = 0$ and $b = 1$) lead to conflicting assignments on stem a. Independent of its initial state upon power-up, the fault-free circuit can never produce the response $\{d, b2, c2\} = \{1, 1, 0\}$. However, if the faulty circuit powers up in state $\{b, c\} = \{1, 0\}$, this output response will be observed. Therefore $c1$ *s-a*-1 is partially testable and hence irredundant under the unrestricted test strategy. Either the $a = 0$ or the $a = 1$ vector is an initialization sequence of this circuit. It can be verified that the same vector which initializes the fault-free circuit also initializes the faulty circuit and that after initialization, the two circuits cannot be distinguished. Hence $c1$ *s*-*a*-1 is redundant under the restricted test strategy. □

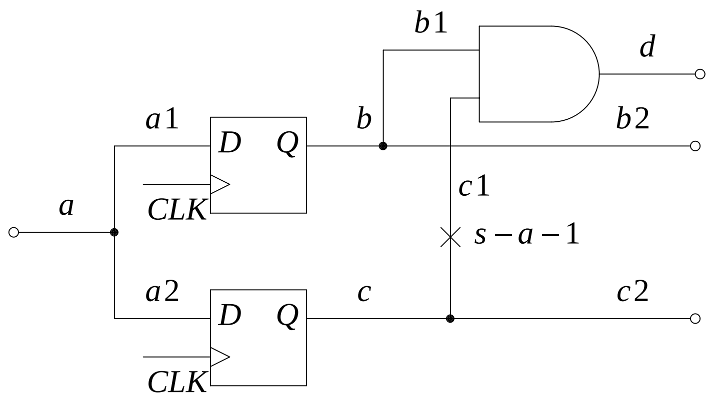


Figure 4. Example of an Irredundant Fault in the Unrestricted Test Strategy

The circuit in Figure 4 is intuitively redundant, since the same signal *a* is fed twice into the same gate *d* through different FFs. Hence it may seem that the restricted test strategy is more practical, since it allows us to identify this redundancy. However, redundancy in the restricted test strategy is not *compositional*. A property of a subcircuit is compositional if that property still holds when the subcircuit is embedded in any larger circuit. When optimizing subcircuits without any knowledge about how the subcircuit will be initialized when it is part of a larger circuit, the restricted test strategy may lead to incorrect results. In such cases, it may be possible for a fault to be redundant in the subcircuit and irredundant in the larger circuit. The following example illustrates this case.

Example 3: Consider the circuit in Figure 5 and the fault $c2$ *s*-*a*-1. The fault is untestable because the requirements to detect it lead to conflicting assignments on stem a. An initialization vector of the fault-free circuit is $\{a, b\} = \{0, 1\}$. The response of the fault-free circuit (independent of its initial state) to the input vector $\{a, b\} = \{0, 1\}$ is $h = 0$. However, for the initial state $\{c, d\} = \{0, 1\}$, the response of the faulty circuit to the same input vector is $h = 1$. Note that this difference in the output responses of the fault-free and the faulty circuits is observed only after the fault-free circuit is initialized. Hence the fault is partially testable and irredundant in either test strategy.

Next, we analyze the subcircuit enclosed by the dashed lines in the figure. An initialization vector for the fault-free subcircuit is $a = 0$. This vector also initializes the faulty subcircuit. After both subcircuits are initialized, they can never be distinguished and hence the fault is redundant in the subcircuit in the restricted test strategy. However, as shown earlier, the fault is irredundant in either test strategy in the larger circuit. □

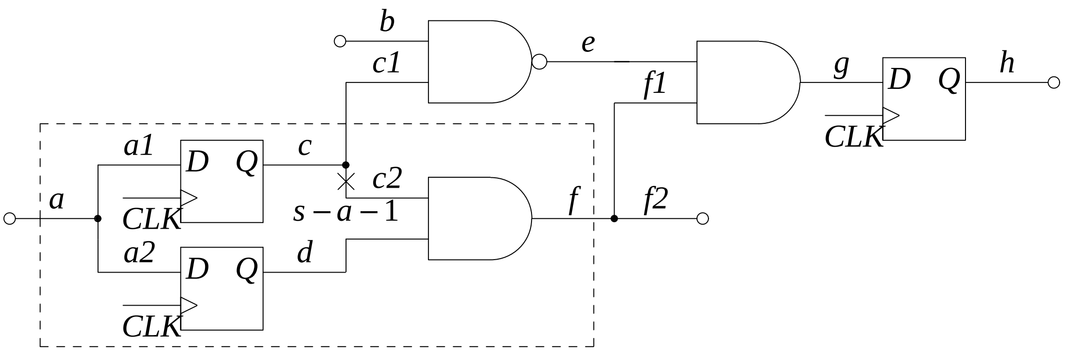


Figure 5. Circuit to Show that Redundancy in Restricted Test Strategy is Not Compositional

It can be shown that redundancy in the unrestricted test strategy is always compositional.

2.3.4 Irredundant Sequentially Untestable Faults. In sequential circuits, untestability and redundancy are not equivalent concepts. One class of sequentially untestable but irredundant faults consists of faults that prevent initialization (FPIs) [Abramovici and Breuer 1979]. A method to identify untestable FPIs was introduced in

[Abramovici and Parikh 1992]. We now present an example of an untestable FPI that is irredundant.

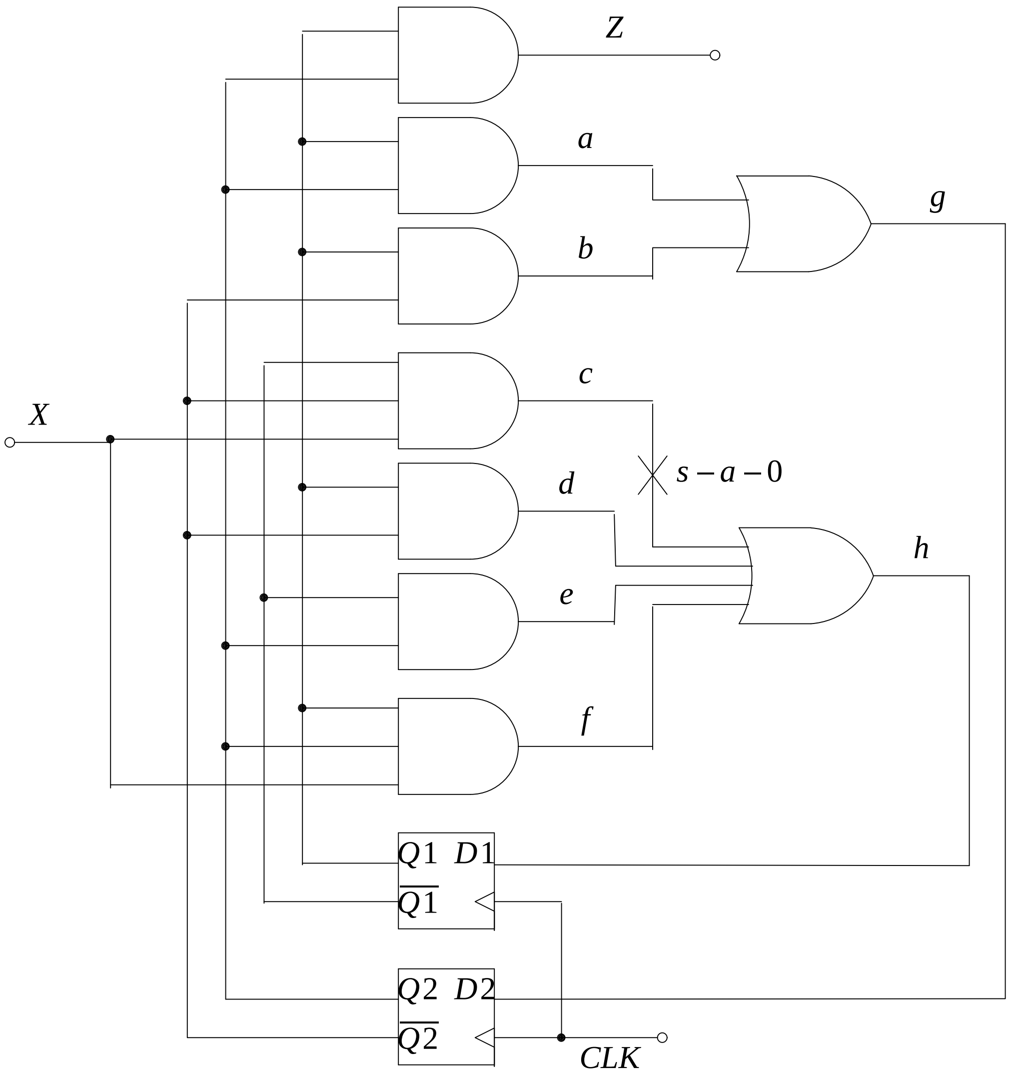
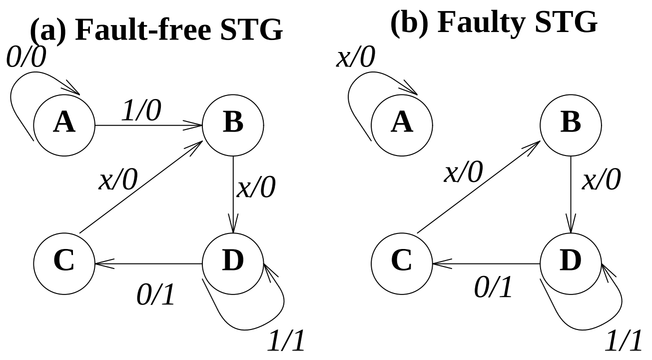


Figure 6. Example Circuit With an Untestable Fault That Prevents Initialization

Example 4: Consider the circuit in Figure 6 and the fault *c s-a-*0. The state transition graphs (STGs) for the fault-free and faulty circuits and the state assignment are shown in Figure 7. The initialization sequence of the fault-free circuit consists of three vectors: $\{0, 1, 1\}$. In the fault-free circuit, state *A* is an illegal state. Any test sequence to activate *c s-a-*0 must drive a 1 onto line *c*, independent of the initial state of the fault-free circuit. This is possible only if state *A* (00) can be reached independent of the initial state of the fault-free circuit. Since in the fault-free circuit, state *A* is illegal, the fault is untestable. However, the input sequence {0, 1, 1, 0} is a partial test for this fault. Independent of its initial state, the output response of the fault-free circuit to this input sequence can never be {0, 0, 0, 0}. However, if the faulty circuit started with state *A*, its response to the input sequence {0, 1, 1, 0} is {0, 0, 0, 0}. Hence the fault is irredundant. It is an FPI because, as shown in Figure 7(b), no initialization sequence exists for the faulty circuit. □



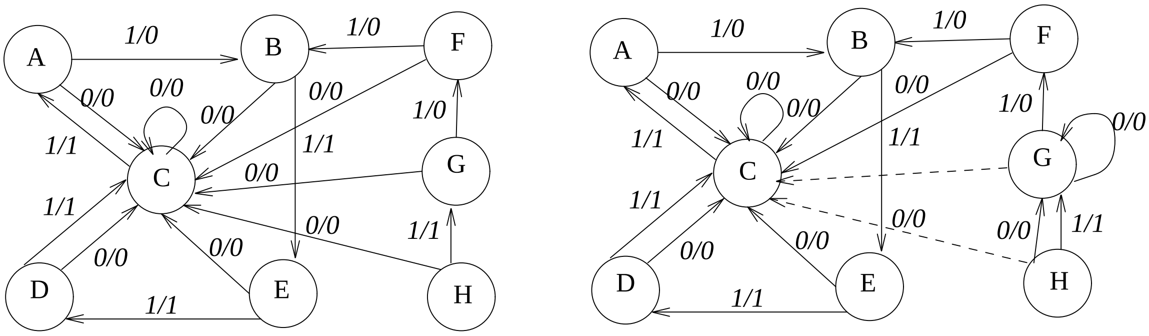
(c) State Assignment

State	*Q1 Q2*
A	*00*
B	*10*
C	*01*
D	*11*

Figure 7. State Transition Graphs for Fault-Free and Faulty Circuit of Figure 6

Cheng conjectured that partial detection may be possible when the fault-free and the faulty circuits are initialized by different sequences [Cheng 1991a, Cheng 1993]. The following example is the first known example to confirm that this conjecture is indeed true and shows that the untestable faults which exhibit this behavior form a new class of untestable but irredundant faults. These are the partially testable faults that modify initialization (FMIs).

Example 5: The fault-free STG shown in Figure 8 has 5 legal states, { *A*, *B*, *C*, *D*, *E* } and 3 illegal states, { *F*, *G*, *H* }. *C* is the reset state. An implementation of this sequential circuit is shown in Figure 9. The state assignment used is { *A*, *B*, *C*, *D*, *E*, *F*, *G*, *H* } = { 000, 001, 010, 011, 100, 101, 110, 111 }. A single



(a) Fault-free STG

(b) Faulty STG

Figure 8. The Effect of an Untestable FMI on the State Transition Graph

vector, $a = 0$, initializes the fault-free circuit to state *C*, with output $z = 0$.

Consider the fault $a1$ s-a-1. This fault modifies the STG as shown in Figure 8(b). The transitions from *H* to *C* and *G* to *C* caused by $a = 0$ are replaced by transitions from *H* to *G* and *G* to *G* respectively. Independent of the initial state of the fault-free circuit, application of the input sequence $\{ 0, 1 \}$ brings the circuit to state *A* and yields the output sequence $\{ 0, 1 \}$. However, if the faulty circuit started from state *H*, application of the same input sequence brings it to state *F* and yields the output sequence $\{ 0, 0 \}$. Thus, $a1$ s-a-1 is partially testable and can be detected only if the faulty circuit starts from state *H*. Since the fault does not affect the legal domain of states and since state *H* is illegal, the fault is untestable. However, the fault is irredundant because the initialization sequence of the fault-free circuit will not work for the faulty circuit. The fault is an FMI because the initialization sequence of the fault-free circuit will not work for the faulty circuit. A longer initialization sequence $(0, 1, 0)$ is required for the faulty circuit. □

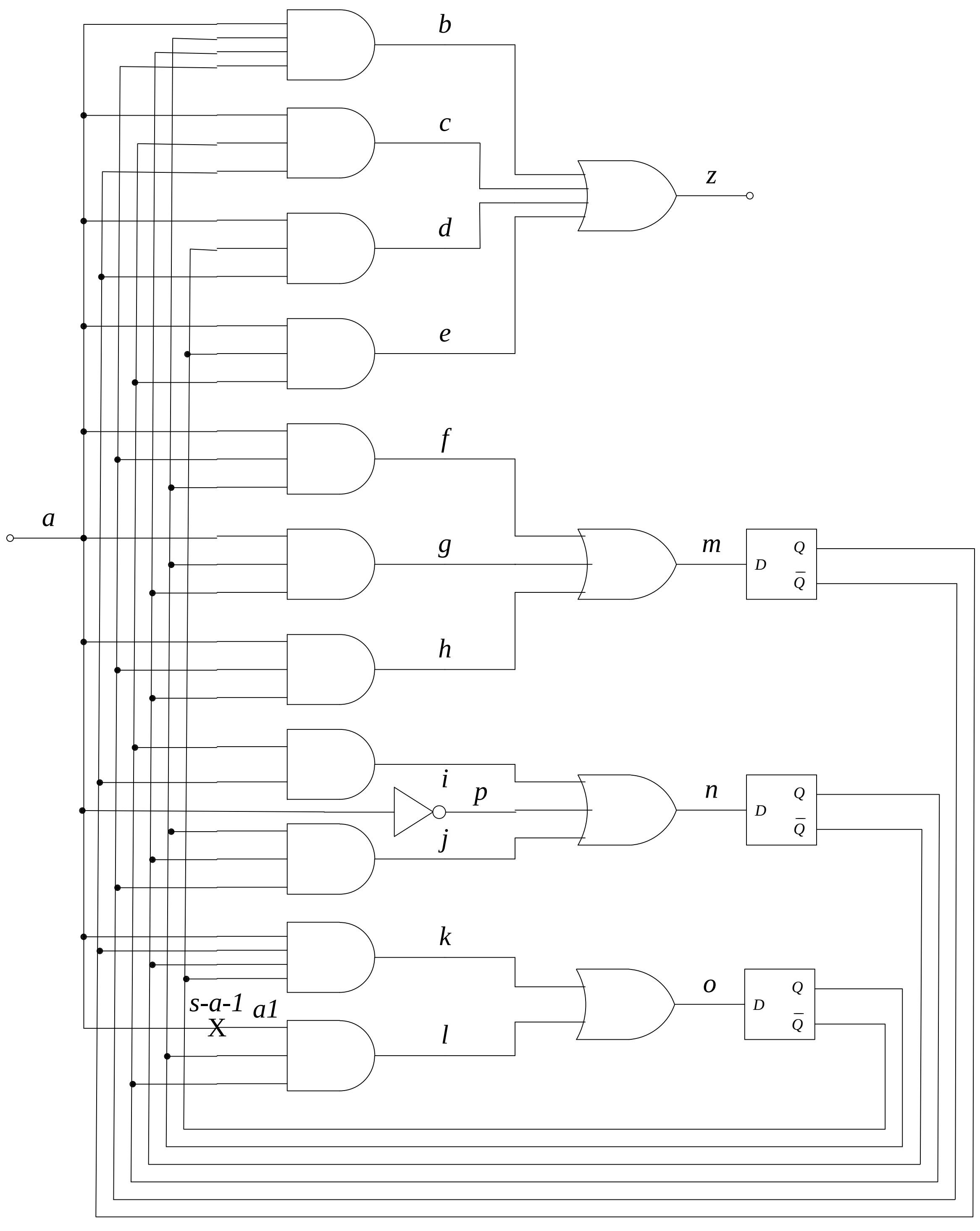


Figure 9. An Example Circuit With a Fault That Modifies Initialization

2.3.5 Sequential Redundancy Removal. Once a sequential redundancy is identified, the procedure to determine (and remove) the redundant logic is similar to the combinational redundancy removal procedure described in Section 2.2. The only difference is that one could remove FFs (in addition to gates) when removing a sequential redundancy. The stuck-value of the redundant fault is forward propagated in the combinational part of the circuit as far as possible. Values can be propagated through FFs only for certain types of FFs. For example, a value v on the input of a simple D FF can be propagated to its output because the output *s-a-v* fault will also be redundant. However, a value $v = 0$ (1) cannot be propagated through a FF that has a preset (reset). In such cases, it is usually recommended to tie that input to a power or ground, as the case may be, rather than propagating the value through the FF. The entire *sequential region* that is bound by the highest level lines to which the redundant stuck-value propagated can be removed from the circuit. Unlike the combinational case, this region could include FFs. A detailed description of the sequential redundancy removal procedure can be found in [Cheng 1991a, Cheng 1993].

CHAPTER III

REVIEW OF PREVIOUS WORK

In this chapter, the existing methods for identifying untestable and redundant faults in logic circuits are reviewed. In sequential circuits, redundant faults form a subset of the sequentially untestable faults. However, combinationally redundant faults are also sequentially redundant. In general, techniques developed for RID and removal in combinational circuits cannot be applied for sequential circuits and hence they are dealt with separately in the following sections. It is shown that some of the solutions for sequential RID are in fact based on incorrect theoretical results [Iyer *et al.* 1996].

3.1 Combinational RID and Removal

RID using ATG is an *indirect* process. Static and dynamic RID techniques are *direct* RID techniques that do not involve ATG-type search. Static RID techniques work as a preprocessing step to ATG, whereas dynamic RID techniques work during ATG. Static RID techniques based on an analysis of reconvergent fanout structures have been proposed in [Harihara and Menon 1989, Menon and Ahuja 1992]. Their analysis requires the repeated computation of *controlling value (c-v) paths* associated with every reconvergent gate of every stem in the circuit. Also, an explicit analysis of dominators is required to claim that a fault is undetectable. A detailed example is shown in Chapter 4 to illustrate their method.

A dynamic RID technique was proposed in [Abramovici *et al.* 1992] that relies on test covering relations among faults and allows the identification of additional redundant faults after the test generator fails to generate a test for a target fault. Their techniques extend beyond fanout-free regions, which are regions without fanouts.

Accelerated RID using ATG has been proposed in [Giraldi and Bushnell 1991]. Their method relies on exploring a reduced ATG search space, by learning previously-encountered equivalent search states. Other existing RID techniques are based on controllability/observability analysis [Ratiu *et al.* 1982] or on structural transformations of the circuit [Abramovici *et al.* 1986].

As stated earlier, redundancy removal has the additional complication that new redundancies may appear as a result of a redundancy removal. It is imperative that any algorithm for redundancy removal identify these redundancies as well. Various techniques have been proposed in the literature to solve the combinational redundancy removal problem.

The method proposed in [Bryan *et al.* 1989] is a "brute-force" method and works as follows. After a test generator completes its task, one of the faults identified as redundant is selected and the circuit is simplified. The new circuit is fault simulated using the vectors currently generated and the remaining undetected faults become the target set for the next test generation run. The entire process is repeated until no more redundancies are found. This approach is inefficient, as it repeats the entire fault simulation/test generation job every time the circuit is simplified, causing many faults to be repeatedly targeted.

Another method for combinational RID and removal has been proposed in [Menon and Ahuja 1992]. Their approach does not use ATG to identify redundancies and they remove a redundant region as soon as it is identified. Their techniques use the analysis of reconvergent fanout structures to identify newly redundant faults after the removal of a particular redundant fault. However, they do not guarantee an irredundant circuit.

Combinational RID and removal based on ATG was proposed in [Kajihara *et al.* 1992]. Their method relies on classification of faults and removing redundancies of only particular classes that are guaranteed not to create new redundancies. This represents removal of only a subset of redundancies. The remaining redundancies are removed by using repeated runs of test generation and fault simulation.

More recently, Chakradhar *et al.* [Chakradhar *et al.* 1995] have proposed a redundancy removal procedure for combinational circuits with non-boolean primitives. Their method separates the function of a non-boolean primitive into boolean and non-boolean components. Only the boolean component is used by an energy-minimization-based test generation algorithm. Their important result is that untestability in such a relaxed formulation of the test generation problem implies redundancy. Their method uses repeated passes of test generation and fault simulation.

In essence, most of these techniques require large amounts of wasted computations. An initial contribution of this research was a novel algorithm for redundancy identification and removal in combinational circuits that uses only a single pass of test generation and fault simulation [Iyer 1991, Abramovici and Iyer 1992]. The important contribution was the development of a novel dynamic RID technique that identifies newly created redundancies that appear as a result of a redundancy removal. This technique was based on analyzing combinations of values that become *illegal* in the new circuit after a redundancy is removed. A general method was proposed to find faults for which a given combination of values (on a set of lines) is a necessary condition for their detection. The redundancies found were not restricted to any particular classes of faults.

3.2 Sequential RID and Removal

In sequential circuits, untestability and redundancy are not equivalent concepts [Abramovici and Breuer 1979]. Typically, irredundant faults that are sequentially untestable affect the initialization of the circuit. Several solutions have been proposed in the literature to identify (and remove) sequential redundancy. In the following sections these solutions are analyzed and it is shown that some of these solutions are based on incorrect theoretical results.

3.2.1 Incorrect Solutions to Identify Sequential Redundancy

3.2.1.1 The Method of Cheng [Cheng 1991b, Cheng 1993]. Cheng addressed the problem of identifying and removing sequential redundancies without assuming a global reset mechanism. He proposed methods to identify special classes of untestable faults that are guaranteed to be redundant. He showed that in a balanced pipeline circuit, every untestable fault is redundant, and that in a general sequential circuit, unactivatable and unpropagatable faults are redundant [Cheng 1991b, Cheng 1993]. (A balanced pipeline circuit is a feedback-free circuit such that, for any signal *s* in the circuit, every path from any primary input to *s* has the same number of FFs.) He devised special procedures to identify these subsets of redundant faults.

The following example illustrates that an unactivatable fault is not necessarily redundant.

Example 6: Consider again the circuit and fault in Figure 6 from Chapter 2. Fault *c s-a-*0 can be activated only by reaching state *A*, independent of the initial states of both circuits. As discussed in Example 4, this is impossible because state *A* is illegal. Hence this fault is an unactivatable fault and *c* always assumes the value 0 after initialization. (This is often referred to as a constant function.) However, as shown in Example 4 and Figure 7 (Chapter 2), the fault is partially testable and irredundant. Cheng's method [Cheng 1991b, Cheng 1993] will classify this fault as redundant and remove its region from the circuit. But this changes the behavior of the circuit, since the new circuit is uninitializable. □

The next example shows that an untestable fault in a balanced pipeline circuit is not necessarily redundant.

Example 7: Consider the balanced pipeline circuit in Figure 10 and the fault $b2$ *s-a-*1. The fault is untestable because its detection requires $b = 0$ and $c = 1$, which causes a conflict on stem a. However, the fault is irredundant under either test strategy. The initialization sequence of the fault-free circuit consists of a single input vector $a = 0$ or $a = 1$. Note that this vector also initializes the output FF g to 0. Independent of its initial state, the response of the fault-free circuit to this input vector is $g = 0$. However, if the faulty circuit started with state $\{b, c\} = \{0, 1\}$, the output response to the input vector $a = 0$ (or $a = 1$) is $g = 1$. Hence $b2$ *s-a*-1 is partially testable and irredundant. □

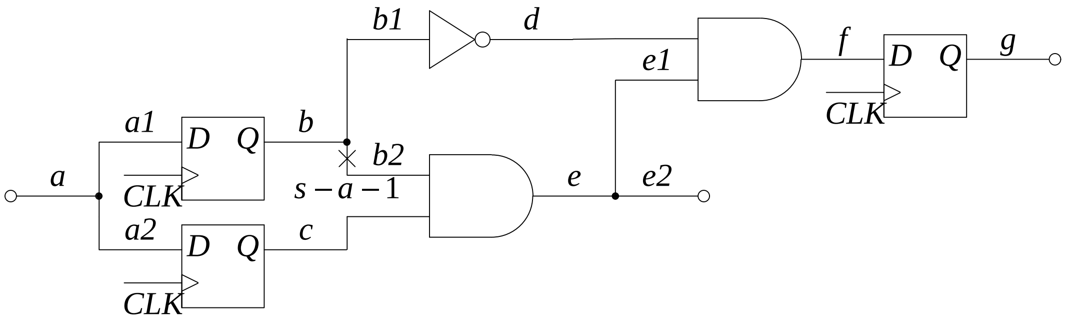


Figure 10. Balanced Pipeline Circuit With an Untestable and Irredundant Fault

3.2.1.2 The Method of Entrena and Cheng [Entrena and Cheng 1993]. The sequential optimization method of Entrena and Cheng [Entrena and Cheng 1993] identifies a redundant fault when the mandatory assignments required to detect it cannot be consistently justified. Their analysis is performed only in the fault-free circuit and they claim a fault as redundant if the mandatory assignments to detect it either end up with conflicts or if the state justification procedure starts looping over previously unjustified states. It will now be shown that both these claims are incorrect. It was also shown in [Pomeranz and Reddy 1994b] that the second claim is incorrect.

Example 8: Consider the circuit in Figure 11 and the fault $f2$ *s-a*-0. The fault is untestable because the requirements in the fault-free circuit to detect it lead to a conflict on stem a. Hence the method of Entrena and Cheng [Entrena and Cheng 1993] will claim this fault as redundant. However, $f2$ *s-a*-0 is irredundant under either test strategy for the following reason. An initialization vector of the fault-free circuit is $\{a,\, b\} = \{0,\, 1\}$. The response of the fault-free circuit to this vector is $i = 1$. However, for the initial state $\{c,\, d\} = \{1,\, 0\}$ of the faulty circuit, its response to the same input vector is $i = 0$. Hence $f2$ *s*-*a*-0 is partially testable. □

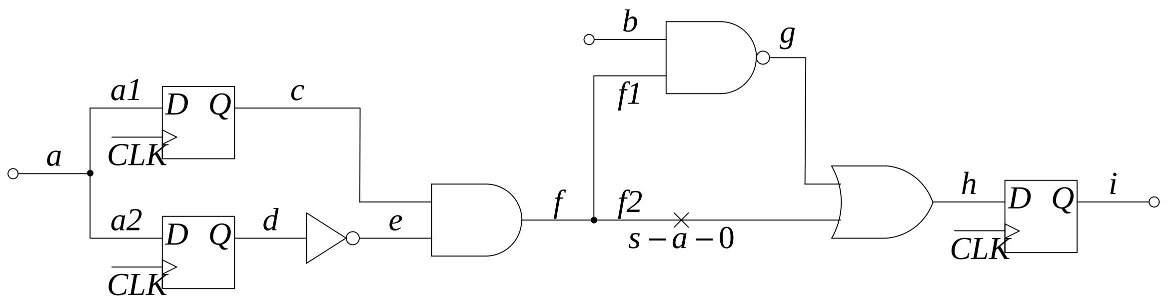


Figure 11. Circuit for Example 8

Example 9: Consider again the circuit in Figure 6 (Chapter 2). Justification of the mandatory assignments to detect this fault requires justifying the state A (00) in both circuits. As shown by the fault-free STG in Figure 7 (Chapter 2), state *A* in the fault-free circuit can only be justified by reaching state *A*. Clearly the fault is untestable. The method of Entrena and Cheng [Entrena and Cheng 1993] will claim it as redundant because the state justification procedure in the fault-free circuit starts looping over the previously unjustified state *A*. However, as shown in Example 4, the fault is partially testable and irredundant. The faulty circuit is uninitializable. □

3.2.1.3 The Method of Liang *et al.* [Liang *et al.* 1994]. Recently, Liang *et al.* [Liang *et al.* 1994] have proposed a method to identify sequentially redundant faults. Their method relies on identifying FFs that cannot be initialized and circuit lines that are uncontrollable. However, their method relies on three-valued logic simulation and this may lead to erroneous results. Their method initially assumes all FFs to be in the unknown *U* state and all PIs to be fully controllable. Simulation is then performed till a stable condition is reached; i.e. the inputs and outputs have the same values for each FF. Then FFs with *U* value are identified as uninitializable. This is similar to infinity values in controllability cost computations [Rutman 1972, Breuer 1978]. The simulation also identifies uncontrollable lines. The following example shows that the method of Liang *et al.* may identify even a testable fault as redundant.

Example 10: Consider the circuit in Figure 12. The method in [Liang *et al.* 1994] will erroneously identify the fault *g s-a*-0 as redundant, when in fact this fault is testable. When simulation is performed (by assuming FF *c* to be in the *U* state and the PIs *a* and *b* to be fully controllable) and it stabilizes, the method will conclude that FF *c* is uninitializable and line *g* cannot be controlled to value 1. This is because *e* and *f* are uncontrollable to value 1 if *c* is uninitializable. Thus, *g s-a*-0 will be classified as redundant, when in fact the vector $\{a, b\} = \{1, 1\}$ is a test for this fault. This vector will produce $g = 1$ independent of the initial state of FF *c*. This is also a fault that will be incorrectly reported as untestable by any test generator using a single unknown logic value. □

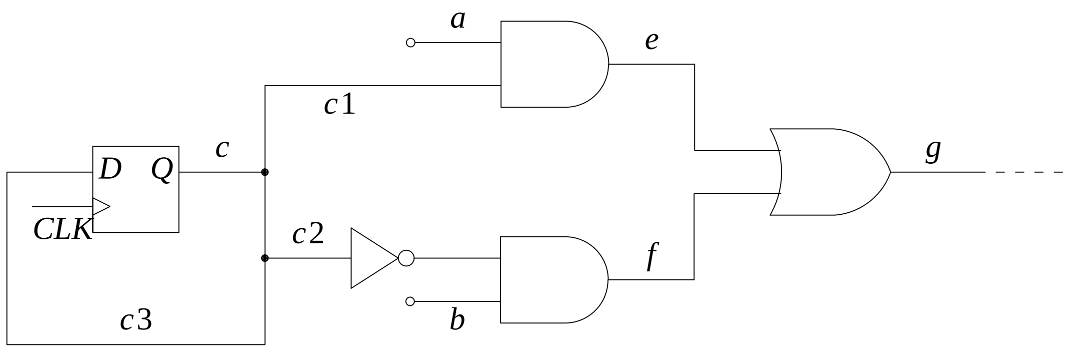


Figure 12. Circuit for Example 10

3.2.2 Limitations of Other Solutions to Identify Sequential Redundancy and Untestability

3.2.2.1 The Method of Cheng [Cheng 1991a, Cheng 1993]. Cheng addressed the problem of identifying and removing sequential redundancies without assuming a global reset mechanism. He proposed a procedure to identify a subset of sequential redundancies (called feedback-free sequential redundancies). The method relies on converting a sequential circuit into a feedback-free model and uses a modified sequential test generator to identify sequentially untestable and redundant faults. However, the procedure used to verify whether an untestable faults is indeed redundant was highly complex and not applicable to large circuits.

3.2.2.2 The Assumption of a Fault-Free Global Reset Mechanism. Cho *et al.* proposed a sequential RID technique based on implicit state enumeration [Cho *et al.* 1993c]. The method relies on the simplifying assumption that the circuit has a fault-free global reset mechanism. The algorithm uses an implicit representation of the STG and determines states that are not reachable from the reset state. This information is used by an ATG algorithm to identify sequential redundancies. The approach uses binary decision diagrams (BDDs) as a platform to deal with the reachability information of sequential circuits. Although this procedure is attractive for small circuits, it cannot handle large circuits.

The methods of [Cho *et al.* 1993c, Devadas *et al.* 1990] assume a fault-free global reset mechanism. Using this assumption, all untestable faults are considered redundant. We now show that this assumption is incomplete.

Example 11: Consider again the circuit and fault *c s-a-*0 in Figure 6 (Chapter 2). As shown in Examples 4 and 6 , this fault is untestable but irredundant. Now let us examine what happens when a fault-free global reset mechanism is added to this circuit. Without loss of generality, we will assume that the reset state is state *D*. The fault-free global reset mechanism can be implemented by adding an extra PI to the OR gates *g* and *h*. Let this new PI be *R*. The STGs of both circuits are now modified so that state *D* can be reached from any state by applying $R = 1$. State *A* still remains illegal in both circuits and the fault is untestable. Since the global reset mechanism is assumed to be fault-free, the faulty circuit can also be initialized. However, if we analyze the circuit by treating R as a regular PI that can be arbitrarily set to a 0 or 1, we can easily see that the test sequence $\{X, R\} = \{ (0, 0), (1, 0), (1, 0), (0, 0)\}$ is a partial test for c *s-a-*0. Hence the fault is still irredundant under Definition 12. □

The results of [Cho *et al.* 1993c] with respect to RID will be correct only if it is also assumed that the operation of the circuit will *always* be started by activating the global reset.

Adding a global reset mechanism to a circuit improves its controllability. Since the circuit with the global reset is more controllable, it may seem that a fault redundant in this circuit (with the additional constraint that the circuit's operation should always be started by using this global reset) should also be redundant in the original circuit without the global reset. Example 11 also illustrates that this is not necessarily true. Thus, adding a global reset mechanism can create a redundancy. More importantly, the method of [Cho *et al.* 1993c] or any other method that identifies sequential redundancies by assuming a fault-free global reset mechanism (which is always used to start the operation of the circuit), cannot be used to identify redundancies in a circuit without a fault-free global reset mechanism.

3.2.2.3 The Methods of Pomeranz and Reddy [Pomeranz and Reddy 1994a, Pomeranz and Reddy 1994b]. Pomeranz and Reddy have proposed a method to identify sequential redundancy, based on combinational ATG for multiple faults, which is not practical for large circuits [Pomeranz and Reddy 1994a]. More recently, they proposed a method to determine whether a sequentially untestable fault prevents the initialization of the circuit [Pomeranz and Reddy 1994b]. They argue that faults which do not exhibit this behavior can be considered as redundant and the circuit simplified accordingly. The method relies on verifying that the faulty circuit has an initialization sequence and requires knowledge of state transition information; thus this method does not seem practical for large circuits. More importantly, their method assumes that the initialization sequence of the circuit can be altered. From a designer's viewpoint this assumption may be unacceptable, since the reset sequence is considered part of the behavior of the circuit. We illustrate this with an example.

Example 12: Consider again the circuit, fault and STGs in Figures 8 and 9 (Chapter 2). The fault was shown to be a partially testable FMI in Example 5. The method of Pomeranz and Reddy [Pomeranz and Reddy 1994b] will consider this fault as redundant and remove its region from the circuit. The original (fault-free) circuit required only one initialization vector $\{a = 0\}$. This vector will not initialize the new (simplified) circuit. In fact, an initialization sequence of three vectors ($a = 010$) is required for the new circuit. □

The other limitation of their method is that it is not compositional; i.e. if we are analyzing a subcircuit of a larger circuit, then it is not safe to remove a fault in a subcircuit without first analyzing the full circuit to see whether the whole system will still have a reset sequence. This is because during initialization some inputs to the subcircuit will come from the larger circuit and some outputs from the subcircuit will be observed by the larger circuit. Thus, checking that the subcircuit still has an initialization sequence is not sufficient.

3.2.2.4 Sequential RID Using Verification. Formal verification techniques were used for sequential RID in [Moondanos and Abraham 1992]. Their method uses Definition 13 and relies on verifying that the fault-free and the faulty circuit (for a redundant fault) have equivalent state tables. This definition of sequential redundancy is rather restrictive. Though their work is of theoretical importance, it is not practical for large circuits, because it relies on building the STGs for the fault-free and faulty circuits for each modeled fault.

3.2.2.5 Retiming and Redundancy. Circuit retiming [Leiserson and Saxe 1991] consists of changing the position of FFs in the circuit. Dey and Chakradhar observed that retiming can be used to convert sequential reconvergences to combinational reconvergences, and that in the process sequentially redundant faults become combinationally redundant [Dey and Chakradhar 1994]. In the following example, we show that retiming can also transform some irredundant sequentially untestable faults into combinationally redundant faults; in other words, retiming synchronous circuitry can create redundancies.

Example 13: Consider again the circuit in Figure 5 (Chapter 2) and the fault *c*2 *s*-*a*-1. As was shown in Example 3, the fault is untestable and irredundant. The fault is untestable because of a conflict caused by the sequential reconvergence of stem *a* at gate *f*. However, forward or backward retiming (see Figures 13 and 14) can be used to convert this sequential reconvergence into a combinational one. The fault *c*2 *s*-*a*-1 then becomes combinationally redundant in the retimed circuit. □

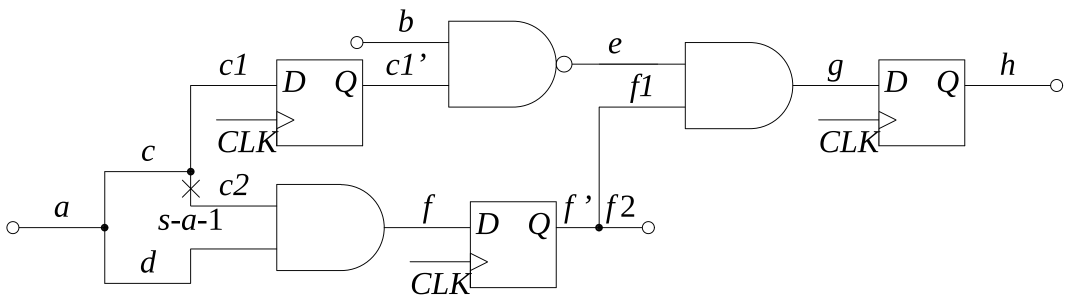


Figure 13. Circuit of Figure 5 After Forward Retiming

Recently, it has been shown that retiming preserves testability [Marchok *et al.* 1994, El-Maleh *et al.* 1994]. Thus an untestable fault in a retimed circuit will also be untestable in the original circuit. The above example illustrates that this is not necessarily true with respect to redundancy.

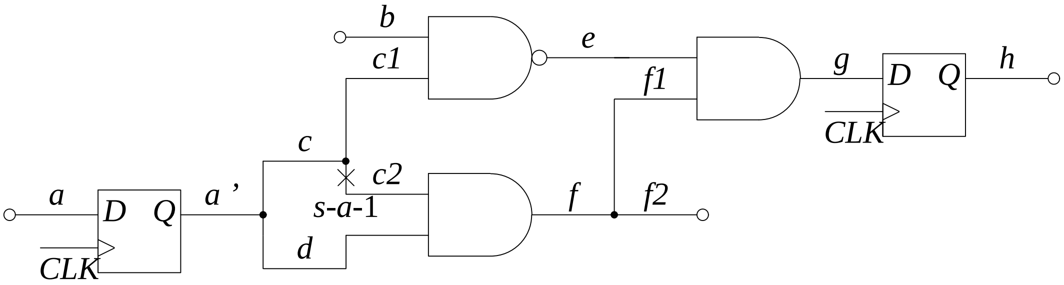


Figure 14. Circuit of Figure 5 After Backward Retiming

3.2.2.6 The Combinational ATPG Theorems of Agrawal and Chakradhar [Agrawal and Chakradhar 1993]. Agrawal and Chakradhar [Agrawal and Chakradhar 1993] proposed two theorems to identify sequentially untestable faults using less complex combinational techniques. Their method uses combinational ATG to target certain faults in an iterative array of finite length derived from a sequential circuit. Their important result is that a combinationally untestable target fault (in the iterative array model of finite length) corresponds to a sequentially untestable fault. Their first (second) theorem, called the single-fault (multi-fault) theorem targets single (multiple) faults in an iterative array model of finite length derived from the sequential circuit. Their method, however, still requires exhaustive search to identify untestability.

It was later shown in [Pomeranz and Reddy 1994a] that untestable faults found using the single-fault theorem are not necessarily redundant. They also showed that all the faults identified using the multi-fault theorem are redundant. However, this procedure requires combinational ATG for multiple faults, which is not practical for large circuits.

All the examples presented in the previous sections suggest that there exists no practical solution for sequential RID. In the following chapters, new algorithms will be presented for identifying sequential redundancies and sequentially untestable faults. These algorithms do not use the search process involved during ATG, but identify a large number of redundant or untestable faults.

CHAPTER IV

FAULT-INDEPENDENT REDUNDANCY IDENTIFICATION IN COMBINATIONAL CIRCUITS

An initial contribution of this research was a novel algorithm for RID and removal in combinational circuits that uses only a single pass of test generation and fault simulation [Iyer 1991, Abramovici and Iyer 1992]. The important contribution was the development of a novel dynamic RID technique that identifies newly created redundancies that appear as a result of a redundancy removal. This technique was based on analyzing combinations of values that become *illegal* in the new circuit after a redundancy is removed. A general method was proposed to find faults for which a given combination of values (on a set of lines) is a necessary condition for their detection.

This chapter presents a novel Fault-Independent algorithm for REdundancy identification (FIRE) in combinational circuits [Iyer and Abramovici 1994a, Iyer and Abramovici 1994b]. FIRE assumes combinational circuits with AND, NAND, OR, NOR and NOT gates as primitives and identifies redundant faults for which a conflict on a single line in the circuit is necessary for their detection. This approach is radically different from any ATG-based approach for RID which identifies a fault as redundant, if all possible ways to detect that fault end up with conflicts. In contrast, FIRE starts with a possible conflict and identifies faults for which that conflict is necessary for detection. Its key advantages are that RID can be accomplished without any search and several redundant faults may be identified by analyzing the same conflict. FIRE has provably polynomial-time complexity. However, since the RID problem is NP-complete [Ibarra and Sahni 1975], FIRE is not guaranteed to identify all redundancies in the circuit. Experimental results on benchmark and real circuits indicate that a large number of redundancies are found, much faster when compared to an ATG-based approach for RID.

4.1 RID Using Illegal Combinations of Values

Assume that $\{X = a,\ Y = b,\ Z = c\}$ is an illegal (impossible) combination of values in the fault-free circuit. Faults for which this combination of values is necessary for detection are untestable and hence redundant. The problem of finding such faults is decomposed, by finding in turn the faults for which each condition is *individually* necessary for detection. Let S_X, S_Y and S_Z be the sets of faults that require $X = a$, $Y = b$, and $Z = c$, respectively, for detection. Then the faults that require $X = a$ and $Y = b$ and $Z = c$ for detection are in the set $S_{XYZ} = S_X \cap S_Y \cap S_Z$. The faults in S_X become untestable when X cannot assume value a (or when X is uncontrollable for value a). These faults are found using the uncontrollability and unobservability analysis introduced in [Abramovici *et al.* 1991].

Controllability is the ability to establish a specific signal value at each node in a circuit by setting values on the circuit's inputs. *Observability* is the ability to determine the signal value at any node in a circuit by controlling the circuit's inputs and observing its outputs. A fault which cannot be activated is said to be uncontrollable whereas a fault which can be activated but cannot be propagated is said to be unobservable. Consequently, any fault which is uncontrollable and/or unobservable can be identified as redundant.

In the uncontrollability and unobservability analysis, $\overline{0}(\overline{1})$ denotes the status of a line that is uncontrollable for value 0(1). For example, to compute the set S_X, the algorithm propagates $X = \overline{a}$ and determines the faults that become untestable if X cannot have the value a; that is, $X = a$ is a necessary condition for the detection of these faults.

Figure 15 illustrates the propagation rules of the uncontrollability indicators. Uncontrollability can propagate forward and backward, as shown by the arrows. For

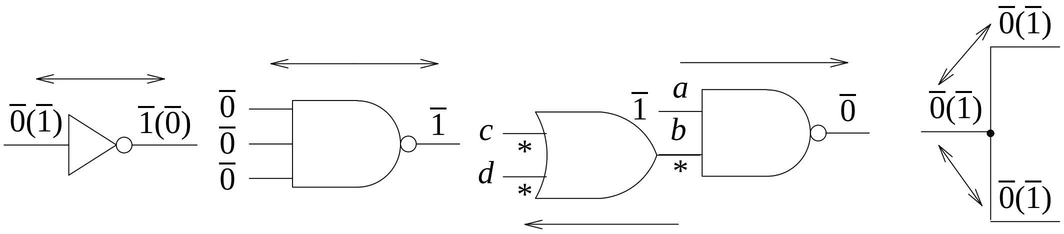


Figure 15. Propagation of Uncontrollability and Unobservability

example, the output of a NAND gate is $\overline{0}$ if at least one input is $\overline{1}$, and is $\overline{1}$ iff all its inputs are $\overline{0}$. Similar rules apply to other gate types. In a combinational circuit, if a gate input cannot be set to the noncontrolling value of the gate, all the other inputs become unobservable. For example in Figure 15, $a = \overline{1}$ implies that b is unobservable (unobservable lines are marked with a "*"). The unobservability status propagates from a gate output backwards to all its inputs; in Figure 15, b being unobservable makes c and d unobservable as well. The meaning of these implications is that $a = 1$ is necessary to observe fault effects from b, c and d. When all fanout branches (FOBs) of a stem are marked as unobservable, the stem is marked as unobservable only if some conditions are satisfied. This is because a stem may be observable by a phenomenon called *multiple-path sensitization*, even if all its FOBs are unobservable. The following example illustrates such a case.

Example 14: Consider the circuit in Figure 16. $a = \overline{1}$ implies $a1 = a2 = b = \overline{1}$. $a1 = \overline{1}$ makes $a2$ unobservable, and $a2 = \overline{1}$ makes $a1$ unobservable. However, a is observable by multiple-path sensitization. □

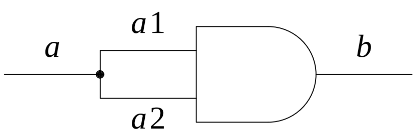


Figure 16. Example of Multiple-Path Sensitization

When all FOBs of a stem are marked as unobservable, the following lemma is used to decide whether the stem can also be marked as unobservable.

Lemma 1: A stem s with all its FOBs marked as unobservable can also be marked as unobservable if for each FOB f of s, there exists at least one set of lines $\{l_f\}$ such that the following conditions are satisfied:

1. f is unobservable because of uncontrollability indicators on every line in $\{l_f\}$; and
2. every line in $\{l_f\}$ is unreachable from s.

Proof: Figure 17 is used for the proof. Assume that a value $\bar{v}$ from some line l implies $a = \bar{1}$ and $b = \bar{0}$, and that a and b are unreachable from s. $a = \bar{1}$ makes $f1$ unobservable and $b = \bar{0}$ makes $f2$ unobservable. To observe stem s, at least one of $\{ a = 1, b = 0 \}$ must be achieved. (Both a and b are unreachable from s, and a fault-effect from s cannot propagate on either of them.) This is possible iff $l = v$. Hence $l = v$ is necessary for the detection of s. □

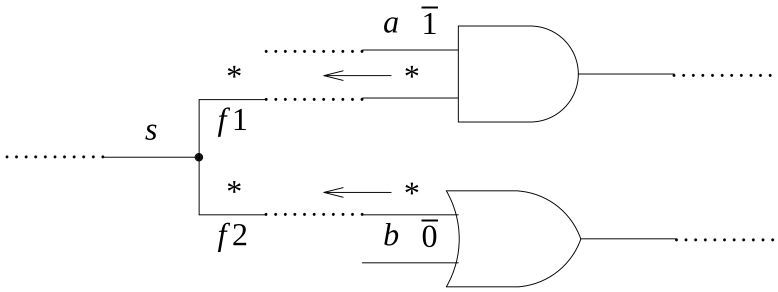


Figure 17. Conditions for Marking Stem Unobservability

While determining uncontrollable and unobservable lines, the faults that become untestable are also identified. These are the faults that cannot be activated (*s-a*-0 on $\bar{1}$ lines and *s-a*-1 on $\bar{0}$ lines) and the faults that cannot be propagated (both faults on unobservable lines). The conditions for which untestable faults that cannot be activated are identified are obvious. However, identifying untestable faults on unobservable lines requires the following justification. The requirements for observing fault-effects may be either in the faulty or the fault-free circuit. Consider for example, the AND gate in Figure 18. To observe a $0/1$ fault-effect (fault-free value/faulty value) on *a* at a PO, a 1

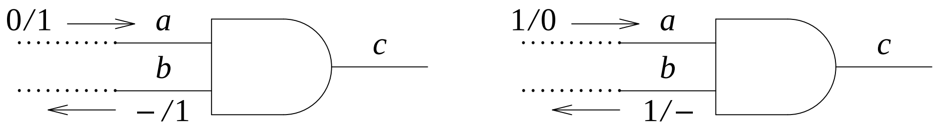


Figure 18. Necessary Conditions for Fault-Effect Observation

must be justified on *b* in the faulty circuit. However, to observe a $1/0$ fault-effect on *a*, a 1 must be justified on *b* in the fault-free circuit. If *b* had a $\overline{1}$, our rules for unobservability propagation claim that a $1/1$ on *b* is necessary to observe both fault-effects on *a*. This is true if any vector (to observe a fault-effect on *a*) which justifies a 1 on *b*, in either the fault-free or the faulty circuit, will drive a $1/1$ on *b*. This condition will be invalidated only if multiple fault-effects arrive at the the inputs of this gate. Lemma 1 in fact pessimistically ensures that this situation does not occur.

This process of propagating uncontrollability and unobservability to determine untestable faults is referred to as *implication*. In summary, the procedure to find the redundant faults caused by an illegal combination $\{l_1 = v_1, \ l_2 = v_2, \ \ldots\}$ is :

1. For every $l_i = v_i$, imply $l_i = \overline{v_i}$ to determine all lines becoming uncontrollable or unobservable. Let S_i be the set of the corresponding faults. (Note that the implications are done separately for each value.)

2. The redundant faults caused by the given illegal combination are in the set $S = \bigcap_i S_i$.

This is a general procedure to find faults for which a given combination of values is a necessary condition for detection. If the combination is illegal, then the faults so found are redundant. The following example illustrates this procedure.

Example 15: Consider the two subcircuits shown in Figure 19. Assume that the function of the subcircuit CKT1 precludes the combination of values $a = 1$, $b = 0$ and $c = 1$. The implications of propagating uncontrollability and unobservability and determining the redundant faults are summarized in Table 1. (A *s-a-v* fault on a line *l* is represented as l_v, and a collapsed set of faults based on fault equivalence is used.) The faults in the set S_a, for example, are those which require $a = 1$ as a necessary condition for their detection.

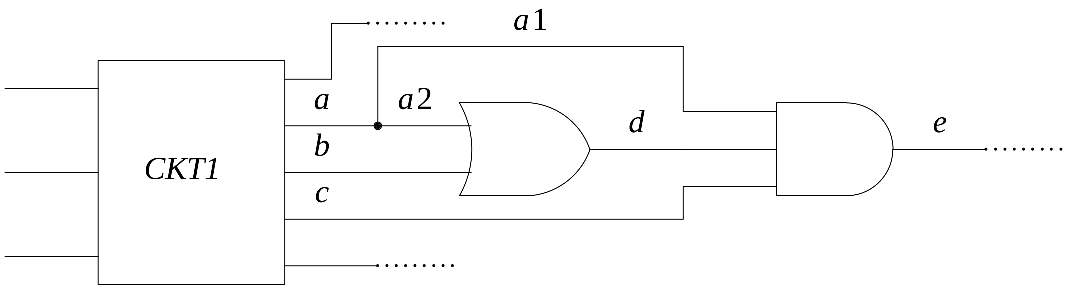


Figure 19. An Example Circuit

Table 1. Implications for Example 15

Process	Uncontrollability	Unobservability	Collapsed Faults
$a = \overline{1}$	$a = a1 = a2 = e = \overline{1}$	$d, a2, b, c$	$S_a = \{a_0, a2_0, d_1, b_0, c_1, e_0\}$
$b = \overline{0}$	$b = d = \overline{0}$	$a2$	$S_b = \{d_1, a2_0\}$
$c = \overline{1}$	$c = e = \overline{1}$	$d, a2, b, a1, a$	$S_c = \{d_1, b_0, a2_0, a1_1, a_0, a_1, e_0\}$
Redundant faults: $S_a \cap S_b \cap S_c = \{d_1, a2_0\}$			

Note that when $c = \overline{1}$, both $a1$ and $a2$ are unobservable and hence a is also unobservable. (c is unreachable from a and $c = 1$ is necessary to observe a.) The information that e is a dominator for a is thus implicitly computed. Thus, $S_a \cap S_b \cap S_c = \{d_1, a2_0\}$, which identifies d *s-a*-1 and $a2$ *s-a*-0 as redundant. It could be verified that $a = 1$, $b = 0$ and $c = 1$ is a necessary condition for the detection of these faults. □

4.2 A Novel Extension for Static RID

The previous section outlined a procedure to identify redundant faults by processing an illegal combination of values on a set of lines, $\{l_1 = v_1, l_2 = v_2, ..., l_n = v_n\}$. The novel extension proposed here is to process *conflicting* values on the same line, *l*, considering $l = 0$ and $l = 1$ as a special case of an illegal combination.

In principle, the analysis could be performed for every signal in the circuit. Relating untestability to reconvergent fanout structures, the signal leads where conflicts occur are fanout stems. The following example illustrates how redundant faults can be identified by processing conflicts on stems.

Example 16: Consider the circuit in Figure 20. A conflict on stem *c* is processed by propagating $c = \overline{0}$ and $c = \overline{1}$ and determining the faults that require $c = 1$ and $c = 0$, respectively, as necessary conditions for detection. The implications of these propagations are summarized in Table 2. The redundant faults identified are $\{ i_0, f_0, g_0, b_0, d_0, a_1 \}$. □

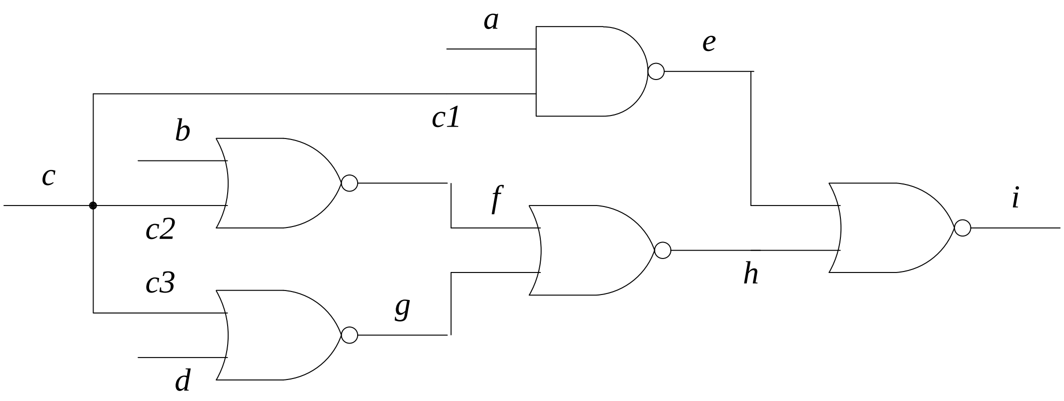


Figure 20. An Example Circuit

In general, a fault could be redundant because of a conflict on at least one stem from a group of stems. The detection of such a fault does not require a conflict on any single

Table 2. Implications for Example 16

Process	Uncontrollability	Unobservability	Collapsed Faults
$c = \overline{0}$	$c = c1 = c2 = c3 = \overline{0}$ $f = g = \overline{1}$ $h = \overline{0}$, $i = \overline{1}$	$b, d, e, a, c1$	$S_0 = \{c1_1, f_0, g_0, i_0, b_0, d_0, e_0, a_1\}$
$c = \overline{1}$	$c1 = c2 = c3 = \overline{1}$ $e = \overline{0}$, $i = \overline{1}$	$a, h, f, g, b, c2, c3, d$	$S_1 = \{ c2_0, c3_0, i_0, h_0, f_0, g_0, b_0, d_0, a_1\}$

Redundant faults: $S_0 \cap S_1 = \{ i_0, f_0, g_0, b_0, d_0, a_1\}$

stem as a necessary condition, and it will not be identified by the above analysis. The observation is that such conflicts often map as conflicts on reconvergent inputs of reconvergent gates.

Example 17: Consider the circuit in Figure 21 and the fault *f s-a*-0. Clearly, the detection of this fault requires $c = d = a1 = b1 = a = b = 1$ as a necessary condition. Also, to propagate the fault-effect from *f*, *e* has to be 0 which requires *either a or b* to have value 0. Thus *f s-a*-0 requires a conflict on either *a* or *b* and processing either *a* or *b* individually for conflicts will not find this fault as redundant. However, as summarized in Table 3, processing a conflict on line *f* (a reconvergent input of a reconvergent gate common to both stems) identifies *f s-a*-0 as redundant.

□

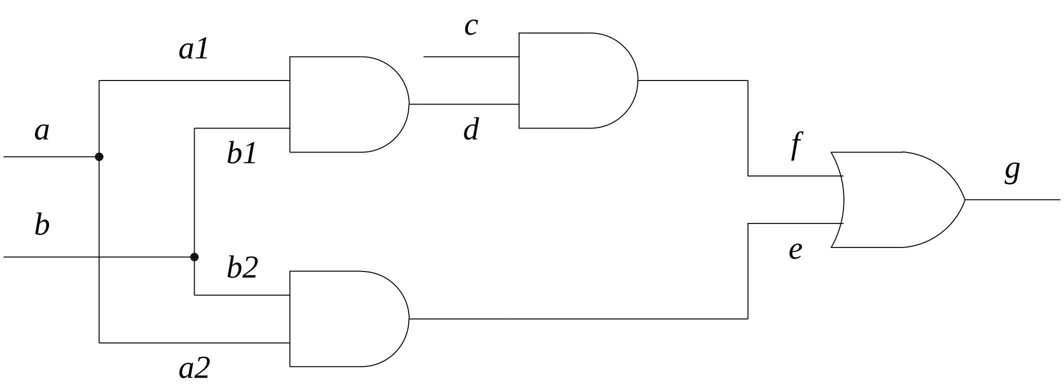


Figure 21. An Example Circuit [Harihara and Menon 1989]

Table 3. Implications for Example 17

Process	Uncontrollability	Unobservability	Collapsed Faults
$f = \overline{0}$	$c = d = a1 = b1 = a = b = a2 = b2 = e = g = \overline{0}$	$f, c, d, a1, b1, e, b2, a2$	$S_0 = \{ c_1, d_1, a1_1, b1_1, a_1, a2_1, b_1, b2_1, g_1, f_0, e_0 \}$
$f = \overline{1}$	$f = \overline{1}$	-	$S_1 = \{ f_0 \}$
Redundant faults: $S_0 \cap S_1 = \{ f_0 \}$			

The preceding examples show that the entire analysis to determine redundant faults is fault-independent and analyzing a single conflict can identify several redundant faults. The new Fault-Independent REdundancy identification algorithm (FIRE) is outlined in Figure 22. Although, in general, processing conflicts on reconvergent inputs of reconvergent gates in addition to fanout stems can find more redundant faults, our algorithm processes conflicts only on fanout stems. This follows from our *empirical observation* that processing conflicts only on stems finds most of the redundancies in the circuit and saves a lot of computation time. All the results reported in Section 4.4 were obtained by processing only stem conflicts.

FIRE() {

For every stem s **in the circuit {**

Imply $s = \overline{0}$ to determine all lines becoming uncontrollable or unobservable.
Let S_0 be the set of the corresponding faults

Imply $s = \overline{1}$ to determine all lines becoming uncontrollable or unobservable.
Let S_1 be the set of the corresponding faults

The redundant faults are in the set $S = S_0 \cap S_1$

}

}

Figure 22. Outline of the FIRE Algorithm

A Note on the Time Complexity of FIRE

Let *n* be the number of lines in the circuit. The worst-case for FIRE would occur when all *n* lines in the circuit are analyzed for conflicts, and implications for uncontrollability and unobservability propagate through the entire circuit. Hence, the worst-case complexity of FIRE is $O(n^2)$. However, in practice: (i) only stems are analyzed for conflicts, and (ii) the implications propagate only through a small subcircuit surrounding the analyzed stem. Therefore the worst-case situation never occurs in practice. Thus, in practice, FIRE finds redundancies with time complexity much lower than $O(n^2)$ and this is very important, considering that the RID problem is NP-complete [Ibarra and Sahni 1975]. However, FIRE is not guaranteed to find all the redundancies in the circuit. The experimental results reported in Section 4.4 attest the efficiency of the algorithm.

4.3 A Complete Example

Consider the circuit in Figure 23 [Harihara and Menon 1989]. It has one reconvergent stem, *d* and three reconvergent inputs *h*, *k* and l1. FIRE will process each of these lines in turn for both 0 and 1 values. The implications of propagating uncontrollability and unobservability and determining the redundant faults for which a conflict on stem *d* is a necessary condition for detection is summarized in Table 4. It can be verified that processing lines *h*, *k* and *l* for conflicts do not find any more redundancies.

The following paragraphs illustrate how these redundancies would be found using the method proposed in [Harihara and Menon 1989, Menon and Ahuja 1992]. To describe their method, a few definitions are reviewed.

- *Definition 14:* An input *i* of a gate *A* is said to be *sensitive* if a change in the signal value at *i* will result in a change in the gate output value.

- *Definition 15:* Let *c* be the controlling value for a reconvergent gate, *G* and let *s* be a stem of *G*. If the value v_s at the stem *s* implies $v_i = c$ at an input *i* of *G*, any path

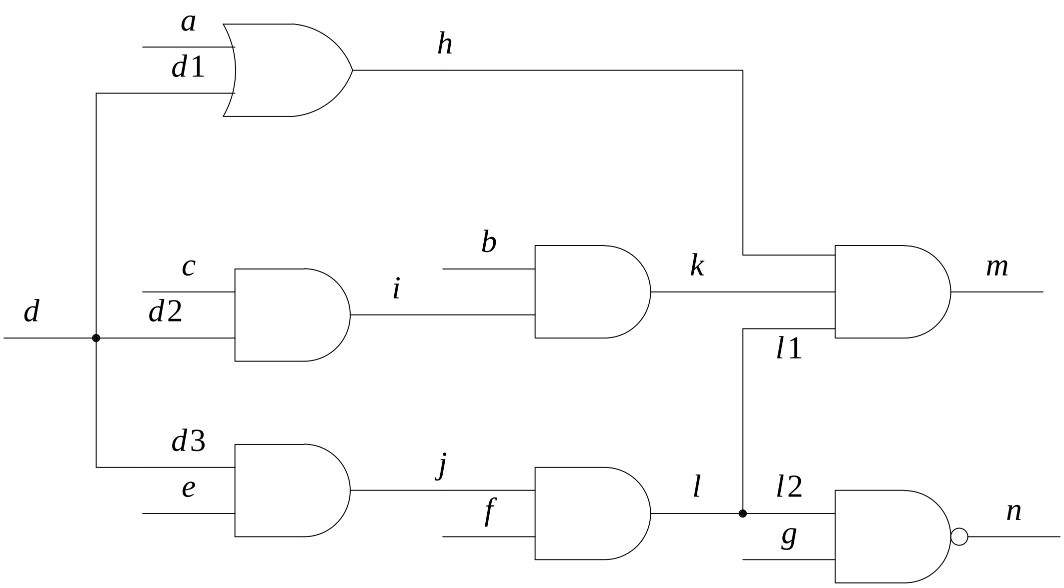


Figure 23. An Example Circuit [Harihara and Menon 1989]

Table 4. Implications for Example of Section 4.3

Process	**Uncontrollability**	**Unobservability**	**Collapsed Faults**
$d = \overline{0}$	$d1 = h = d2 = d3 = \overline{0}$	a	$S_0 = \{ d_1, d2_1, d3_1, h_1, a_0 \}$
$d = \overline{1}$	$d1 = d2 = d3 = i = j = k = l = l1 = l2 = m = \overline{1}, n = \overline{0}$	$c, e, b, f, h, d1, a, l1, k, i, d2, g$	$S_1 = \{d_0, d1_0, m_0, l_0, n_1, c_1, e_1, b_1, f_1, h_1, a_0, l1_1, k_1, i_1, d2_1, g_1 \}$

Redundant faults: $S_0 \cap S_1 = \{ a_0, d2_1, h_1 \}$

from *s* to *i*, obtained by tracing back from *i* along sensitive inputs until *s* is reached is called a *controlling value path* or *cv path* for the stem value v_s. Paths from *s* to *G* without the above property will be called *non-cv paths* for the stem value v_s.

The method essentially consists of finding *cv paths* by assigning both values (0 and 1) to every stem and tracing back from every reconvergent gate to the stem. Setting $d = 1$ implies $d1 = d2 = d3 = h = 1$. Thus, no cv paths are produced for this stem value. Setting $d = 0$ implies $d1 = d2 = d3 = i = k = m = j = l = l1 = l2 = 0$ and

$n = 1$. The cv paths produced for this implication are $\{ d2, i, k \}$, $\{ d3, j, l, l1 \}$ and $\{ d3, j, l, l2 \}$. Also, $d = 0$ is a necessary condition for the detection of d_1, $d1_1$, $d2_1$, $d3_1$, a_0, a_1 and h_1. There are two reconvergent gates *m* and *n* for stem *d*. Each of these gates is analyzed separately. The analysis is shown in detail for the reconvergent gate *m*. Similar conditions can be verified for gate *n*.

Since FOBs $d1$ and $d2$ are bound by gate *m*, $d1$ *s-a*-1 and $d2$ *s-a*-1 are identified as untestable. This is because $d = 0$ is a necessary condition for the detection of these faults and there exist cv paths from *d* to *m* not passing through their sites. Also $d = 0$ is necessary to propagate any fault effect from *a* through gate *h* and hence *a s-a*-0 and *a s-a*-1 are untestable. Next gate *n* is analyzed with its cv paths from stem *d* and this does not find any untestable faults. Note that for any fault on $d3$ to be untestable, it has to be proven untestable at both its dominators *m* and *n*. However, this analysis finds $d3$ *s-a*-1 to be untestable only at *m*, and hence this fault is not claimed to be untestable.

Clearly, the computations required for this method are more intensive when compared to FIRE. This is because every reconvergent gate of every stem is analyzed *explicitly* for cv paths and a fault has to be shown as untestable at each of its dominators for it to be redundant.

4.4 Experimental Results

FIRE was implemented in C and was run on the ISCAS85 combinational benchmark circuits [Brglez and Fujiwara 1985] and on the full-scan versions of the ISCAS89 sequential benchmark circuits [Brglez *et al.* 1989]. The production version of FIRE uses special models for FFs in a circuit. Table 5 summarizes the characteristics of the benchmark circuits after expansion of the FF models. *Gates*, *Lines*, *Flip-flops*, *PIs*, *POs*, and *Faults* represent the number of gates, lines, FFs, PIs, POs and faults respectively in the corresponding circuit. Note that the number of gates for the sequential benchmark circuits are higher than those contained in the original benchmark

Table 5. Circuit Parameters for Benchmark Circuits

Circuit Name	Gates	Lines	Flip-flops	PIs	POs	Faults
C1908	880	1908	0	33	25	1879
C2670	1193	2594	0	157	64	2595
C3540	1669	3540	0	50	22	3428
C5315	2307	5315	0	178	123	5350
C6288	2416	6288	0	32	32	7744
C7552	3512	7551	0	206	107	7548
S208	168	313	8	12	2	217
S298	245	481	14	4	6	310
S344	304	540	15	10	11	344
S349	305	545	15	10	11	352
S382	347	656	21	4	6	401
S386	213	465	6	8	7	386
S400	351	674	21	4	6	426
S420	340	629	16	20	2	432
S444	370	718	21	4	6	476
S510	265	589	6	20	7	566
S526	382	800	21	4	6	557
S526n	383	800	21	4	6	555
S641	551	888	19	36	24	469
S713	564	961	19	36	23	583
S820	334	886	5	19	19	852
S832	332	898	5	19	19	872
S838	678	1255	32	36	2	859
S953	610	1239	29	17	23	1081
S1196	689	1427	18	15	14	1244
S1238	668	1469	18	15	14	1357
S1423	1323	1386	74	18	5	1517
S1488	707	1567	6	9	19	1488
S1494	701	1573	6	9	19	1508
S5378	4390	7623	179	36	49	4605
S9234	7649	12199	228	20	22	6929
S13207	13972	21877	669	32	121	9817
S15850	15145	23609	597	15	87	11727
S35932	31937	58397	1728	36	320	39096
S38417	36903	59608	1636	29	106	31182
S38584	32321	57309	1452	13	278	36305

circuits. This increase is due to the gates internal to the FF models. The number of faults are those obtained after functional fault equivalence collapsing [Abramovici *et al.* 1990]. Faults internal to FF models are however ignored. For all the sequential benchmark circuits, a clock PI and both its faults are also included.

To compare the savings in computations for FIRE with respect to a fault-oriented approach for RID, the LTG test generation/ fault simulation package [Abramovici *et al.* 1986] was used Faults proved redundant by FIRE were passed as targets to LTG with a backtrack limit of 100. LTG also has the advantage of dynamic RID techniques [Abramovici *et al.* 1992]. It is true that in a free run, LTG would, in general, identify more redundancies than FIRE. However, the purpose of these experiments is to testify how much time LTG would save by not targeting the faults proved redundant by FIRE.

Table 6. FIRE Results for Benchmark Circuits

Circuit Name	FIRE # Red.	FIRE CPU sec.	LTG (Bt. Limit=100) # Red.	LTG (Bt. Limit=100) # Abo.	LTG (Bt. Limit=100) CPU sec.	Speed-up Ratio
C1908	6	1.8	6	0	5.0	2.8
C2670	29	1.5	29	0	3.3	2.2
C3540	93	11.9	93	0	9.7	0.8
C5315	20	2.8	20	0	10.0	3.6
C6288	33	1.3	33	0	25.9	19.9
C7552	30	4.7	30	0	17.3	3.7
S349	2	0.4	2	0	0.7	1.8
S444	11	0.2	8	0	0.7	3.5
S713	32	0.1	32	0	1.0	10.0
S1238	6	1.9	6	0	1.6	0.8
S1423	5	0.3	5	0	2.6	8.7
S1494	1	1.1	1	0	1.4	1.3
S5378	34	3.7	34	0	15.8	4.3
S9234	165	20.6	158	7	55.6	2.7
S13207	55	23.2	55	0	30.4	1.3
S15850	234	20.5	234	0	157.8	7.7
S35932	3984	235.6	3984	0	2502.4	10.6
S38417	99	45.4	99	0	275.2	6.1
S38584	1037	156.0	1036	0	1264.7	8.1

Table 6 shows the comparison of RID in benchmark circuits between FIRE and LTG. These results are for a collapsed set of faults. We show only the the circuits where redundancies were identified by FIRE. *#Red.* represents the number of identified redundancies. The *CPU times* are in seconds for a SUN sparc2. *#Abo.* represents the number of targets aborted by LTG. Speed-up ratio represents (LTG time)/(FIRE time).

Table 7. Circuit Parameters for Real Circuits

Circuit Name	Gates	Lines	Flip-flops	PIs	POs	Faults
Test7	1020	2114	31	10	7	1891
Amd2910	1709	3553	87	21	16	2575
Chip17	1439	3015	22	8	17	2954
Chip58	1654	3105	83	42	45	2191
Test132	2224	5075	24	138	64	5070
Test117	3040	6611	82	28	59	5839
Test149	4073	8545	88	88	99	7955
Test198	4414	9783	87	61	54	8949
Test105	5741	11932	162	69	181	10567
Test154	5913	13051	108	144	138	12104
Test122	6896	16350	1	273	305	16866
Test66	7288	16462	162	31	163	14840
Test119	12795	29561	183	229	235	28575
C42825	18507	41648	0	1179	1180	39037

Table 8. FIRE Results for Real Circuits

Circuit Name	FIRE		LTG			Speed -up Ratio
	# Red.	CPU sec.	# Red.	# Abo.	CPU sec.	
Test7	94	4.6	94	0	7.3	1.6
Amd2910	102	4.5	102	0	12.3	2.7
Chip17	113	2.3	113	0	22.1	9.6
Chip58	6	2.2	6	0	2.6	1.2
Test132	336	4.7	336	0	158.7	33.8
Test117	295	5.8	295	0	217.0	37.4
Test149	674	38.3	666	8	454.9	11.9
Test198	396	35.8	396	0	268.6	7.5
Test105	737	29.8	737	0	615.3	20.7
Test154	517	41.1	517	0	410.4	10.0
Test122	952	145.9	938	14	1431.0	9.8
Test66	305	63.2	305	0	594.2	9.4
Test119	988	215.9	988	0	2432.2	11.3
C42825	2630	1013.0	2630	0	2050.8	2.0

For almost all the circuits, FIRE performed much better than LTG. For S35932, FIRE found 3984 redundant faults in 236 secs. LTG found all of them to be redundant in 2502 secs. Thus, in this case, FIRE is about 11 times faster than LTG. Furthermore, for S9234, LTG could not solve all the target faults identified as redundant by FIRE.

Table 7 summarizes the characteristics of some real circuits and Table 8 shows the FIRE results obtained for these circuits. FIRE performed much better than LTG for all the circuits. A speed-up ratio of up to 37 was achieved for these circuits.

Table 9 shows the percentage of redundancies identified by FIRE in the benchmark circuits. Since TRAN [Chakradhar *et al.* 1993] is a state-of-the-art combinational test generator that reports all the single-fault redundant faults in the benchmark circuits, it was used for comparisons. The TRAN CPU times are also for a SUN sparc2 and include times for random test generation, transitive closure algorithm and fault simulation. Note that TRAN can find more redundancies at the expense of more CPU time. In some cases (like S35932), FIRE did a complete job of finding all the redundancies. However, in some other cases (like S1238) FIRE did not perform as well.

Table 9. Percentage of Redundancies Identified By FIRE

Circuit Name	FIRE		TRAN		% FIRE of Total	% FIRE time of TRAN time
	# Red.	CPU sec.	# Red.	CPU sec.		
C1908	6	1.8	7	13.0	85.7	13.9
C2670	29	1.5	115	95.2	25.2	1.6
C3540	93	11.9	131	24.9	71.0	47.8
C5315	20	2.8	59	32.3	33.9	8.7
C6288	33	1.3	34	38.0	97.1	3.4
C7552	30	4.7	131	308.0	22.9	1.5
S349	2	0.2	2	0.3	100.0	66.7
S444	11	0.2	14	0.4	78.6	50.0
S713	32	0.1	38	3.1	84.2	3.2
S1238	6	1.9	69	17.4	8.7	10.9
S1423	5	0.3	14	8.5	35.7	3.5
S1494	1	1.1	12	3.7	8.3	29.7
S5378	34	3.7	40	73.0	85.0	5.1
S9234	165	20.6	452	803.7	36.5	2.6
S13207	55	23.2	151	806.5	36.4	2.9
S15850	234	20.5	389	1177.5	60.2	1.7
S35932	3984	235.6	3984	1617.0	100.0	14.6
S38417	99	45.4	165	5078.2	60.0	0.9
S38584	1037	156.0	1506	2483.8	68.9	6.3

We believe that the reasons for such a behavior are (i) our implication procedure is not complete; (ii) there could be redundant faults that do not require a single conflict as a necessary condition for detection, and (iii) Lemma 1 to mark unobservability on a stem is pessimistic and provides only a sufficient condition. We also show the percentage of FIRE CPU time of the total CPU time used by TRAN to identify these redundancies. For example, in S38417, FIRE identified 60% of the combinational redundancies in 0.9% of the total time used by TRAN to identify all the redundancies. FIRE found 5876 (about 80%) of the total 7313 redundant faults in these circuits. In summary, FIRE identified between 8.3% and 100% of the redundant faults identified by TRAN and FIRE CPU times ranged from 0.9% to 66.7% of TRAN CPU times. Note that in every case when FIRE time was more than 15% of the TRAN time, the FIRE time was under 12 seconds. The results in Table 9 indicate that FIRE can be an useful preprocessor even to state-of-the-art test generators like TRAN.

The next chapter introduces the concept of *c*-cycle sequential redundancy. The concept of FIRE is extended to identify *c*-cycle sequential redundancies and the sequential FIRE algorithm (FIRES) is presented.

CHAPTER V

IDENTIFYING SEQUENTIAL REDUNDANCIES WITHOUT SEARCH

The previous chapter presented a new fault-independent algorithm for static combinational RID. Faults for which a conflict on a single line in the circuit are necessary for their detection were found to be untestable and hence redundant. Also, a subset of lines in the circuit that need to be analyzed for this purpose were identified.

A sequentially untestable fault is not necessarily redundant [Abramovici and Breuer 1979]. Chapter 2 also illustrated some examples of irredundant sequentially untestable faults. Chapter 3 reviewed all the previous work to identify sequential redundancy and it was shown that no practical solution exists to solve this problem. This chapter introduces the concept of a c-cycle sequential redundancy [Iyer *et al.* 1995]. A fault is considered as redundant if the behavior of the faulty circuit is indistinguishable from that of fault-free circuit after applying c arbitrary input vectors to the faulty circuit. c-cycle redundancy is a generalization of the conventional notion of sequential redundancy which is represented by the particular case $c = 0$. The approach of FIRE is extended to identify c-cycle sequential redundancy. The important result is that a fault which requires a conflict as a necessary condition for its detection is c-cycle redundant. The sequential FIRE algorithm (FIRES) also identifies the value of c for every c-cycle redundancy it identifies.

An intermediate result of this research was an algorithm to Find sequentially UNTESTable faults (FUNTEST) without search [Iyer and Abramovici 1994c], that extended the concept of FIRE to sequential circuits using the single-fault theorem in [Agrawal and Chakradhar 1993]. However, the use of the single-fault theorem precludes FUNTEST from identifying sequential redundancy. Even though FIRES uses an implication mechanism similar to FUNTEST, it is significantly different. FIRES identifies c-cycle redundancies, whereas FUNTEST can identify only sequentially untestable faults. The theoretical basis for FIRES is different from that of FUNTEST; FIRES is based on conflict analysis whereas FUNTEST is based on the single-fault theorem of [Agrawal and Chakradhar 1993]. It will also be shown that FIRES can identify faults that are beyond the scope of the combinational ATPG theorems in [Agrawal and Chakradhar 1993].

5.1 c-Cycle Redundancy

In this section, the concept of a *c*-cycle redundancy is introduced. *c*-cycle redundancy is a generalization of the conventional notion of sequential redundancy. Consider any arbitrary input sequence I. If a fault is redundant according to Definition 12, then for *every* state S^f of the faulty circuit, there must exist some state S of the fault-free circuit such that the response of the two circuits (starting from S^f and S) to I is identical. Now suppose that we power up the faulty circuit and clock it c times, applying arbitrary inputs. After this procedure, the set of possible states that the faulty circuit could be in is usually a proper subset of the set of all states, i.e., values in some of the FFs will become correlated. Suppose that we only require that every S^f in this smaller set have a corresponding state S in the fault-free circuit for which the response to I is identical. Under this condition, we can still use the faulty circuit as a replacement for the fault-free circuit provided that we clock the faulty circuit c times before applying I.

Definition 16: Consider the set of states $\{S_c\}$ reachable after powering up the faulty circuit and applying any sequence of c input vectors. (Note that $\{S_c\}$ shrinks as c increases.) Then a fault is *c*-cycle redundant if it is not partially testable under the assumption that the initial states of the faulty circuit are restricted to $\{S_c\}$.

Note that a 0-cycle redundant fault would be a redundant fault under Definition 12. Also, a *c*-cycle redundant fault is a c'-cycle redundant fault for any $c' > c$. A circuit with a *c*-cycle redundant fault can be simplified by removing the redundant region associated with that fault. A *c*-cycle redundant fault is not the same as an untestable fault that is not an FPI [Pomeranz and Reddy 1994b]. Removing a *c*-cycle redundant fault only requires the application of *c arbitrary* input vectors before the existing initialization sequence. In contrast, removing a fault from the latter category [Pomeranz and Reddy 1994b] requires the computation of a new initialization sequence.

It is reasonable to remove *c*-cycle redundant faults since it is easy to have the circuit clocked a few times before the initialization sequence is applied. In fact, other sequential optimizations may require the same sort of delay before initialization. For example, in recent work, Singhal *et al.* [Singhal *et al.* 1995a] have shown that forward retiming [Leiserson and Saxe 1991] has this same property. Using the terminology of [Singhal *et al.* 1995b], it can be said that removing a *c*-cycle redundancy from a circuit creates a *c*-cycle delayed replacement for the original circuit. The notion of *c*-cycle redundancy also has the nice property of being compositional. Hence, removing a *c*-cycle redundant fault in a subcircuit can be done without examining the larger circuit; we just have to clock the entire system *c* times before applying its usual initialization sequence. In contrast, with methods such as [Pomeranz and Reddy 1994b], it is not safe to remove a fault in a subcircuit without first analyzing the full circuit to see whether the whole system will still have a reset sequence. This is because during initialization some inputs to the subcircuit will come from the larger circuit and some outputs from the subcircuit will be observed by the larger circuit. Thus, checking that the subcircuit still has an initialization sequence is not sufficient.

Example 18: Consider the circuit in Figure 5 (Chapter 2). As was shown in Example 3 (Chapter 2), the fault $c2$ *s-a-*1 is untestable and irredundant. An initialization vector of the fault-free circuit is $\{a, b\} = \{0, 1\}$. Consider applying an arbitrary input vector to the faulty circuit before the input vector $\{a, b\} = \{0, 1\}$. The application of this arbitrary vector ensures that the FFs *c* and *d* have the same value, and makes it impossible to distinguish the faulty circuit from the fault-free circuit. Thus, this fault is a 1-cycle redundant fault. Note that after applying one arbitrary input vector to the faulty circuit, the original initialization sequence still works. □

In the previous example, note that $c2$ *s-a*-1 is not redundant (based on Definition 12) and the circuit cannot be simplified (by removing line c2) based on that definition. But considering it as a 1-cycle redundant fault allows circuit simplification. Hence identification of *c*-cycle redundancy for redundancy removal seems more practical.

5.2 The Algorithm

5.2.1 Sequential Uncontrollability and Unobservability Analysis. Extension of the combinational FIRE algorithm to sequential circuits requires the propagation of uncontrollability and unobservability indicators through FFs. Figure 24 illustrates the propagation rules for FFs. Uncontrollability propagates both ways through a FF. Propagation through a FF means going into an adjacent time frame.

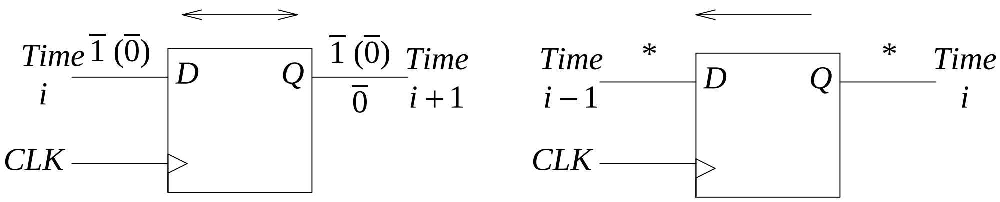


Figure 24. Sequential Propagation of Uncontrollability and Unobservability

Though uncontrollability can propagate forward and backward, unobservability propagates only backward. When a FF output is unobservable, its input is also unobservable. While in a combinational circuit, unobservability propagates unconditionally from the output of a gate to all its inputs, in sequential circuits this is not always the case. Before marking a gate input as unobservable, we ensure that multiple fault-effects from that input in different time frames cannot combine to reach a PO. Like in the combinational case, we must avoid marking faults that would be detected by (sequential) multiple path sensitization. Hence propagation of unobservability in the sequential case is allowed under the following generalization of the combinational conditions.

Unobservability propagates onto l^i (the copy of line l at time i) if

1. The fanouts of l^i are marked as unobservable at time i.
2. For every fanout f^i of l^i, there exists at least one set of lines $\{ p^j \}$, such that
   - f^i is unobservable because of uncontrollability indicators on every line in $\{ p^j \}$; and
   - there is no sequential path from l^k, $i \le k \le j$ to any line in $\{ p^j \}$.

The implementation actually uses a simplified version of this criterion that is easier to check.

5.2.2 Rules for Marking Faults. The iterative array model of a sequential circuit consists of repeated time frames of its combinational logic. For a faulty circuit, the fault is present in every time frame. The uncontrollability and unobservability propagations described in the previous section are valid only for the fault-free circuit. In general, they may not be valid in a faulty circuit. To ensure that these propagations will also be valid in a faulty circuit (for reasons explained later), a validation step is performed as follows.

Let time frame 0 represent the time at which the uncontrollability started propagating from some stem s (see Figure 25). The time frames through which uncontrollability and unobservability propagate have positive or negative numbers depending on whether they are forward or backward time frames with respect to the starting time frame. Let f and b be the number of forward and backward time frames respectively. This propagation is allowed for a maximum of T_M time frames ($f + b + 1 \le T_M$). T_M is needed to prevent an infinite propagation of uncontrollability and unobservability (due to feedback loops). However, many propagations terminate before the total number of frames reaches T_M.

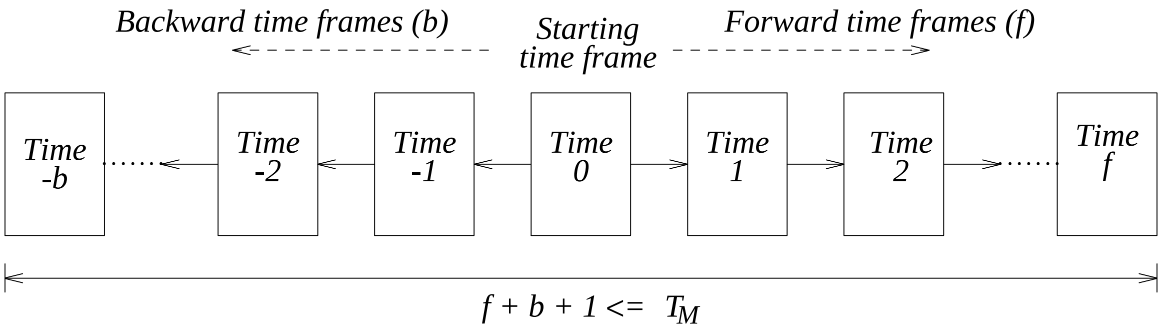


Figure 25. Sequential Implications Over Various Time Frames

Definition 17: A value $\overline{v}$ on a line l^i (a copy of line l in time frame i), $-b \le i \le f$, is valid in the presence of a fault m *s-a-u* iff $\overline{v}$ propagates onto l^i in the presence of the faults m *s-a-u* in all time frames $j < i$.

For example, if it was necessary to have $m = \overline{u}$ in a previous time frame to get $l^i = \overline{v}$, then $l^i = \overline{v}$ is invalid in the presence of m *s-a-u*. Note that the validation of an uncontrollability indicator is with respect to a fault; i.e. an uncontrollability may be valid in the presence of one fault and invalid in the presence of another.

For the stem s and the experiments $s = \overline{v}$, $v \in \{0, 1\}$ and for each time frame i ($-b \le i \le f$), a set of faults S_v^i is formed as follows

- *Uncontrollable faults:* If a line m has a *valid* $\overline{0}$ ($\overline{1}$) value, at time i, m *s-a*-1 (m *s-a*-0) is included in S_v^i.
- *Unobservable faults:* Let a line m have an unobservability indicator at time i caused by uncontrollability indicators on a set of lines $\{p^j\}$ ($j \ge i$). m *s-a-u* ($u \in \{0,1\}$) is included in S_v^i, iff the uncontrollability indicator on every line in $\{p^j\}$ is valid in the presence of the fault m *s-a-u*.

The process of propagating uncontrollability and unobservability to determine the uncontrollable and unobservable faults is referred to as *sequential implication*.

5.2.3 The FIRES Algorithm. FIRES in turn processes every stem s in the circuit and finds faults for which $s = 0$ and $s = 1$ is necessary for detection by

1. sequentially propagating uncontrollability and unobservability for a maximum of T_M time frames and
2. forming sets S_v^i of faults (using the rules above).

Then, the set of faults for which a conflict on s is necessary for detection are determined as $S_s = \bigcup_i \{S^i\}$, where $S^i = S_0^i \cap S_1^i$.

It can be shown that all the faults identified by FIRES without the validation step are untestable. (The proof will be similar to the proof of Theorem 2 in the next chapter.) The reason to validate the uncontrollability implications in the faulty circuit is to prove redundancy and is explained in Appendix A.

Consider a fault f identified by FIRES in some time frame i. Let l be the leftmost time frame where uncontrollability must propagate for FIRES to identify f as redundant. The following rule is used to associate a c_f with every fault f identified by FIRES.

- $c_f = 0$, if $l \geq i$
- $c_f = i - l$, if $l < i$

c_f represents the number of time frames for the conflict to get established; clocking the faulty circuit c_f times in fact ensures that the time frames required for the conflict to occur really exist. These rules may overestimate the value of c_f; i.e. f may actually be c_f'-cycle redundant for some $c_f' < c_f$. To determine the minimum c_f, a more global analysis may be required than that performed by FIRES. A justification for these rules can be found in Appendix A.

Figure 26 summarizes the FIRES algorithm.

FIRES(T_M) {

/* T_M = Maximum number of time frames */

For every stem s

Sequentially Imply $s = \overline{0}$ over a maximum of T_M time frames. Let S_0^i be the sets of the corresponding uncontrollable or unobservable faults.

Sequentially Imply $s = \overline{1}$ over a maximum of T_M time frames. Let S_1^i be the sets of the corresponding uncontrollable or unobservable faults.

Every fault f in any set $S^i = S_0^i \cap S_1^i$ is c_f-cycle redundant.

}

}

Figure 26. Outline of the FIRES Algorithm

Theorem 1: A fault f identified by FIRES is c_f-cycle redundant.

Proof: See Appendix A. □

It is emphasized that the c-cycle redundant faults found by FIRES are not restricted to any particular classes of faults as in [Cheng 1991b, Cheng 1993], or to just faults that cannot be detected in the time frame that they are first activated as was addressed by the single-fault theorem [Agrawal and Chakradhar 1993], or to just faults that cannot be activated and propagated in a finite number of time frames as was addressed by the multi-fault theorem [Agrawal and Chakradhar 1993]. The following example shows how FIRES can find a fault that is beyond the scope of the combinational ATPG theorems [Agrawal and Chakradhar 1993].

Example 19: Consider the circuit in Figure 27. Independent of the number of time frames used, based on the combinational ATPG theorems [Agrawal and Chakradhar 1993] we will never find the faults *e s-a*-1, *a s-a*-0, and *b s-a*-0 as untestable, because these faults can always be detected at a state variable in the rightmost time frame of a finite time frame model. However, these faults can never be propagated to a PO.

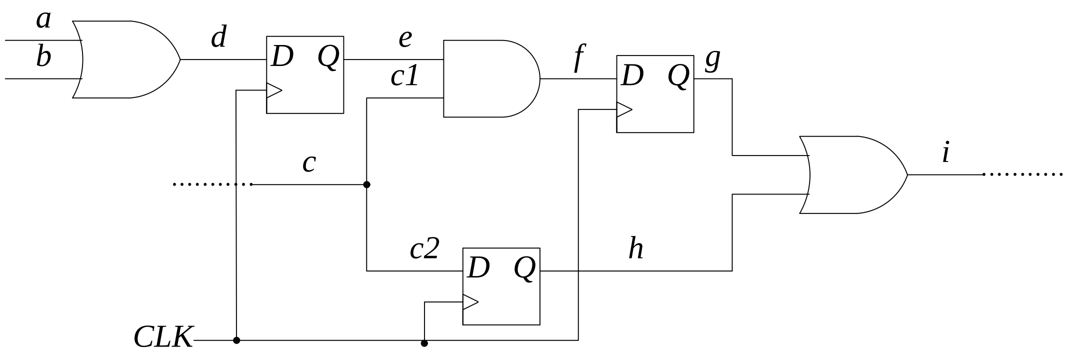


Figure 27. Circuit for Example 19

Table 10. Sequential Implications for Example 19

Process	Time	Uncontrollability	Unobservability	Collapsed Faults
$c = \overline{0}$	0	$c = c1 = c2 = \overline{0}$	$f, e, c1$	$S_0^0 = \{c_1, c1_1, c2_1, f_0, f_1, e_1\}$
	1	$h = i = \overline{0}$	g	$S_0^1 = \{i_1, g_0\}$
	-1	–	d, a, b	$S_0^{-1} = \{d_0, d_1, a_0, b_0\}$
$c = \overline{1}$	0	$c = c1 = c2 = f = \overline{1}$	e	$S_1^0 = \{c_0, c2_0, f_0, e_1\}$
	1	$h = g = i = \overline{1}$	–	$S_1^1 = \{h_0, g_0, i_0\}$
	-1	–	d, a, b	$S_1^{-1} = \{d_0, d_1, a_0, b_0\}$
Redundant Faults	0	$S_0^0 \cap S_1^0 = \{f_0, e_1\}$		
	1	$S_0^1 \cap S_1^1 = \{g_0\}$		
	-1	$S_0^{-1} \cap S_1^{-1} = \{d_0, d_1, a_0, b_0\}$		

FIRES will find these redundant faults as follows. $c = \overline{0}$ at time 0 implies $c1 = c2 = \overline{0}$ at time 0. This implies $h = i = \overline{0}$ at time 1. This makes line *g* at time 1 unobservable, which makes lines *f*, *e* and $c1$ at time 0 and lines *d*, *a* and *b* at time -1 unobservable as well. A similar analysis with $c = \overline{1}$ shows that fault effect on *e* at time 0 and fault effects on *d*, *a* and *b* at time -1 are unobservable. The uncontrollability and unobservability propagations and the sets of faults identified are summarized in Table 10. Note that for each process ($c = \overline{0}$ or $c = \overline{1}$), the sets of faults are maintained separately for the different time frames.

Note that when computing the S_0 and S_1 sets of faults in all the time frames, all uncontrollability implications remain valid in the presence of the faults found in the corresponding time frames. With every fault f identified by FIRES, a c_f is associated such that f is c_f-cycle redundant. By the rules of computing c_f, all the faults except g_0 are 0-cycle redundant. Fault g_0 is 1-cycle redundant. □

Among the seven redundant faults identified by FIRES in the above example, the FUNTEST algorithm [Iyer and Abramovici 1994c] will report only one (g_0) as untestable.

5.3 Implementation and Experimental Results

FIRES was implemented in C++ and productized under a new object-oriented framework that supports circuits with wired structures, tristate devices, bidirectionals, busses, multiple clocks and asynchronous loops. Special models were devised to support these special devices. The tool is technology-independent and supports majority of the combinational components and FFs in any AT&T standard cell library. Means are also provided to support other vendor libraries. The tool also supports technology-independent primitive gates called AND, NAND, OR, and NOR with arbitrary number of inputs and the NOT gate. It also extracts clocking information automatically.

The FIRES tool provides three ways to solve the problem of a component block not being supported.

1. The user can supply the definition of that component in some directory, specified by a library path option.
2. The user can alias the unknown component block to some known component block using an aliasing option.
3. The component block can be treated as an unrecognized device. FIRES temporarily cuts the device from the circuit. The inputs of the device are treated

as POs of the circuit and the outputs of the device are treated as PIs of the circuit. Note that redundant faults found in the new circuit are also redundant in the original circuit.

FIRES was used to identify redundancies in the ISCAS89 sequential benchmark circuits [Brglez *et al.* 1989]. FIRES performs the validation step outlined in Section 5.2.2 as an option. Note that faults found by FIRES without the validation step are untestable, and those found with the validation step are *c*-cycle redundant. In general, without validation, FIRES runs faster and finds more faults.

Table 11. FIRES Results for Benchmark Circuits

Circuit	# Frames	FIRES Without Validation		FIRES With Validation			
		# Unt.	CPU secs.	# Red.	CPU secs.	0-cycle	Max_c
S208	13	60	0.9	57	1.7	0	4
S349	4	2	0.3	2	0.3	2	0
S386	4	27	0.5	27	0.5	0	2
S400	12	1	0.9	1	1.1	0	2
S420	15	217	5.7	208	10.1	0	7
S444	11	11	1.3	11	1.5	11	0
S713	15	32	0.6	32	0.7	32	0
S838	15	529	25.4	508	50.1	0	11
S1238	3	6	2.5	6	2.6	6	0
S1423	10	5	1.2	5	1.3	5	0
S1494	3	1	1.5	1	1.6	1	0
S5378	15	273	37.3	273	46.2	48	11
S9234	15	284	90.2	270	132.2	165	6
S13207	10	661	77.0	661	84.0	67	9
S15850	10	332	312.4	328	385.8	235	9
S35932	5	3984	525.6	3984	545.7	3984	0
S38417	5	147	295.5	147	352.6	115	1
S38584	5	1397	234.4	1397	258.4	1072	3
S499*	15	5	12.2	5	18.6	0	2
S1269*	12	1	0.9	1	1.1	0	2
PROLOG*	5	253	5.9	253	8.7	82	2
S3330*	5	162	6.9	162	8.7	12	1

Table 11 shows the results for the benchmark circuits. *# Frames* represents the number of time frames which FIRES processed. (The maximum number of time frames is decided depending upon the circuit size, such that $0 < \#Frames \leq 15$.) We show FIRES results with and without the validation step. *#Unt.* represents the number of untestable faults identified by FIRES without performing the validation step. *#Red.* represents the number of *c*-cycle redundant faults identified by FIRES (after validation). *0-cycle* represents the number of 0-cycle redundancies identified (these are combinational, and conventional sequential, redundancies). *Max_c* represents the maximum of the c_f values for the faults found by FIRES. All CPU times reported in this section are in seconds for a SUN sparc10. Only those circuits are shown where redundancies were identified by FIRES. The circuits shown with a "*" were obtained from the ADDENDUM93 directory of the ISCAS89 benchmark set. These circuits, except PROLOG, have no single combinational redundancies.

For example in S420, without validation, FIRES found 217 sequentially untestable faults in 5.7 seconds. 208 out of these 217 faults passed the validation and hence are *c*-cycle redundant. None of them were 0-cycle and the maximum *c* was found to be 7. The run with the validation step took about 10 seconds.

The distribution of the number of *c*-cycle redundancies found for different values of *c* varies widely from circuit to circuit. Table 12 summarizes this distribution. As an example of the distribution of *c* values, in S5378, FIRES found 48 *0*-cycle, 29 *1*-cycle, 163 *2*-cycle, 9 *3*-cycle, 12 *4*-cycle, 6 *5*-cycle, 4 *8*-cycle, 1 *9*-cycle, and 1 *11*-cycle redundancies. Note that these *c* values are conservative and that computation of the minimum *c* requires a more global analysis than that performed by FIRES. The "-" in the table indicate that FIRES did not process that many time frames. For example, in S208, FIRES processed 13 time frames (see Table 11). Hence columns *13* and *14* have "-".

Unfortunately, these results cannot be compared with any of the previously published ones. The results of [Cheng 1991a, Cheng 1991b, Cheng 1993, Cho *et al.* 1993c, Pomeranz and Reddy 1994b] are for redundancy removal, those of [Cho *et al.* 1993c] are not relevant for the original benchmark circuits without a global reset, those of [Agrawal and Chakradhar 1993] are only for untestability, those of [Moondanos and Abraham 1992] deal with a restricted definition of redundancy, those of [Cheng 1991a, Cheng 1991b, Cheng 1993, Entrena and Cheng 1993] are based on incorrect theoretical results, and those of [Pomeranz and Reddy 1994a, Pomeranz and Reddy 1994b] do not deal with benchmark circuits.

Table 12. Summary of *c*-cycle Redundancies Found By FIRES

Circuit	*C*-cycle Redundancies														
Name	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S208	0	2	21	0	34	0	0	0	0	0	0	0	0	-	-
S349	2	0	0	0	-	-	-	-	-	-	-	-	-	-	-
S386	0	0	27	0	-	-	-	-	-	-	-	-	-	-	-
S400	0	0	1	0	0	0	0	0	0	0	0	0	-	-	-
S420	0	6	46	3	88	1	55	9	0	0	0	0	0	0	0
S444	11	0	0	0	0	0	0	0	0	0	0	-	-	-	-
S713	32	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S838	0	14	62	11	86	0	62	0	58	57	58	100	0	0	0
S1238	6	0	0	-	-	-	-	-	-	-	-	-	-	-	-
S1423	5	0	0	0	0	0	0	0	0	0	-	-	-	-	-
S1494	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-
S5378	48	29	163	9	12	6	0	0	4	1	0	1	0	0	0
S9234	165	0	33	4	2	0	66	0	0	0	0	0	0	0	0
S13207	67	29	162	53	18	3	5	0	120	204	-	-	-	-	-
S15850	235	40	7	0	39	1	1	0	2	3	-	-	-	-	-
S35932	3984	0	0	0	0	-	-	-	-	-	-	-	-	-	-
S38417	115	32	0	0	0	-	-	-	-	-	-	-	-	-	-
S38584	1072	36	287	2	0	-	-	-	-	-	-	-	-	-	-

A deterministic sequential test generator performing exhaustive search can identify all faults found by FIRES as untestable, if given enough time. However, the test generator cannot identify these faults as redundant. We now show that state-of-the-art sequential test generators can have difficulty even in proving that the faults found by FIRES are untestable. We used two sequential test generators, GENTEST [Chakraborty *et al.* 1991] and HITEC [Niermann and Patel 1991]. In this experiment, the faults found by FIRES (without validation) were passed as the only targets to the test generators. For S5378, GENTEST, allowed to spend up to 100 seconds per fault, aborted 95 % of the untestable faults found by FIRES. Moreover, GENTEST used 45 times more CPU time than FIRES. (See Table 13.)

Table 13. Comparison of FIRES With GENTEST for S5378

Circuit	FIRES		GENTEST			Speed-up
Name	# Unt.	CPU secs.	# Unt.	# Abo.	CPU secs.	Ratio
S5378	273	37.3	15	258	1670.5	45

Similarly, HITEC, allowed to spend up to 2 seconds per fault in the first pass and up to 20 seconds in the second, could prove untestability only for 52% of the faults found by FIRES in S838. These results are summarized in Table 14. These examples show that the exhaustive search done by the sequential test generators could not identify as untestable faults found by FIRES without any search.

Table 14. Comparison of FIRES With HITEC for S838

Circuit	FIRES		HITEC						Speed-up
			Pass 1: 2 secs./fault			Pass 2: 20 secs./fault			
Name	# Unt.	CPU secs.	# Unt.	# Abo.	CPU secs.	# Unt.	# Abo.	CPU secs.	Ratio
S838	529	25.4	235	294	677.1	276	253	6136.8	242

Experience has shown that the FIRES CPU times are negligible when compared to a complete sequential test generation run for all faults in the circuit. This is because FIRES *identifies redundant faults without any search*, whereas any sequential test generator would perform exhaustive search for any untestable fault in the circuit. (However, FIRES is not guaranteed to identify all redundancies in a circuit.) This suggests that using FIRES as a preprocessor to a sequential test generator would give the following benefits:

1. the unique capability of identifying sequential redundancy;
2. improvements in the detectable fault coverage; and
3. reduction in the run-time by not targeting the redundant faults identified by FIRES.

The first point above is the prime benefit from a synthesis point of view and is the main theme of this chapter.

5.4 Other Solutions for Sequential RID

Chapter 3 discussed in detail all the existing solutions to identify sequential redundancy. It was shown how some solutions are incorrect. Now it is shown that some of these solutions in fact identify c-cycle redundancy.

The following example shows that an untestable fault in a balanced pipeline circuit may be c-cycle redundant (and irredundant under Definition 12), $c > 0$.

Example 20: Consider again the circuit and fault in Figure 10. The fault was shown to be untestable and irredundant in Example 7. The fault is irredundant (partially testable) because for the faulty initial state $\{b, c\} = \{0, 1\}$ and an input vector $a = 0$ (or $a = 1$), the response of the fault-free and the faulty circuits are different. Now consider applying an arbitrary input vector to the faulty circuit before applying the initialization vector $a = 0$ (or $a = 1$). The application of this vector ensures that the FFs b and c have the same value, thus ensuring that the faulty circuit and the fault-free circuit are indistinguishable. Thus this fault is a 1-cycle redundant fault. □

Next it is shown that retiming may convert a c-cycle redundant fault into a combinational redundancy.

Example 21: Consider the circuit in Figure 5 and the fault $c2$ *s-a*-1. Example 3 showed that the fault is untestable and irredundant and Example 18 showed that it is a 1-cycle redundant fault. Forward or backward retiming can be used to convert the sequential reconvergence of stem a at gate f into a combinational one (see Example 13 and Figures 13 and 14). The corresponding fault in the retimed circuit then becomes combinationally redundant. □

CHAPTER VI

IDENTIFYING SEQUENTIALLY UNTESTABLE FAULTS USING ILLEGAL STATES

Sequential ATG is a very complex problem. One of the complicating factors is the existence of illegal states. A sequential test generator may waste a lot of time trying to justify illegal states. This often occurs when the target fault is untestable, a situation recognized only when the exhaustive search for a test sequence terminates without finding any. This search process relies on backtracking to recover from dead-end situations. A dead end is characterized by encountering either an inconsistency (conflicting assignments) or a loop of illegal states $s_1, s_2, ..., s_n$, where s_{i+1} is required to justify s_i $(1 \le i < n)$, and s_1 is required to justify s_n.

Clearly, determining illegal states by an efficient technique as a preprocessing step to ATG would significantly reduce its computational complexity. In this chapter, we first review a recently proposed algorithm to Find ILLegal states (FILL) [Long *et al.* 1995] using implicit state enumeration based on binary decision diagrams (BDDs) [Bryant 1986]. Other techniques [Coudert *et al.* 1989, Coudert and Madre 1991, Lin *et al.* 1990, Touati *et al.* 1990, Cho *et al.* 1993b, Cho *et al.* 1993c] require the unrealistic assumption of a (fault-free) reset state. These techniques compute a set of legal states using a forward search; but if this step is not completed because of excessive memory or run-time requirements, then no illegal states are found. The sequential test generator described in [Cho *et al.* 1993a] does not assume a reset state but computes an initialization sequence and performs a forward search from the computed set of reset states. However, it has limited application because it cannot handle large circuits. In contrast, FILL does not assume a reset state, does not compute an initialization sequence, and builds up illegal states incrementally, allowing it to at least obtain partial solutions. Pixley's method [Pixley 1990] is similar to FILL, though it will in general find fewer illegal states. A new functional partitioning method allows FILL to handle large circuits. FILL does not guarantee finding all the illegal states, but the complete set of illegal states is not necessary in ATG or in other applications such as sequential logic optimization.

Next, an algorithm to Find UNtestable faults for which Illegal states are necessary for detection (FUNI) is presented. FUNI uses the illegal states computed by FILL. This is more general than the FUNTEST algorithm [Iyer and Abramovici 1994c], which processes conflicts on stems to identify untestable faults. A conflict is a special case of an illegal combination of values, which assigns opposite values to the same line. It is also different from the FIRES algorithm presented in the previous chapter; FIRES identifies *c*-cycle redundancies based on conflict analysis whereas FUNI identifies untestable faults using illegal states. In general, all these algorithms identify different sets of faults. Unlike ATG, FUNI is fault-independent and does not rely on exhaustive search; rather, it uses an implication procedure similar to FIRES. (Other work to identify sequentially untestable faults without using sequential ATG [Agrawal and Chakradhar 1993] is based on combinational ATG which still requires exhaustive search.)

6.1 Review of the FILL Algorithm [Long *et al.* 1995]

FILL computes illegal states in a sequential circuit without assuming a reset state and without having to compute an initialization sequence. Recall that illegal states are states that are not reachable after initialization of the circuit. (For formal definitions the reader is referred to Chapter 2.) FILL uses BDDs as the basic representation of the circuit. Other methods using BDD-based search techniques to identify illegal states rely on a forward traversal of the state space of the machine starting from a set of reset states. The forward traversal yields the set of legal states, which is then complemented to get the set of illegal states. However, if the reset information is not available or if the forward search must be aborted because the BDD representation of the reached state set grows too large or because the search takes too long due to a high sequential depth, then no illegal states are identified by these methods. FILL does not require reset information, and it builds up illegal states directly, so if the procedure must be terminated it still returns partial information. However, it is not guaranteed to find all the illegal states in the circuit. Pixley's ‘‘outer envelope’’ computation [Pixley 1990] uses the same basic approach of eliminating states that must be illegal, though in general it may find fewer illegal states than FILL. FILL functionally partitions large circuits to simplify the analysis.

The Approach Used By FILL [Long *et al.* 1995]

FILL does not require reset information because of the following observation. If a circuit is initializable, then all the terminal SCCs in its STG are equivalent. All the states in these terminal SCCs are legal states. FILL iteratively eliminates states that cannot be part of any terminal SCC. Initially, all states are considered potentially legal. Next FILL uses the following observation: if state *s* has no potentially legal predecessor states (including *s*), then *s* is illegal.

For example, in Figure 28, state *A* is illegal. Note that this observation does not eliminate state *B*, because *B* has a potentially legal predecessor, namely *B* itself. However, FILL first eliminates self-loops from the STG without changing the set of terminal SCCs. By doing so, any state without incoming edges is either illegal or forms a terminal SCC by itself, so in Figure 28, FILL sees that *B* is illegal. FILL uses this observation repeatedly to prune the set of potentially legal states. FILL may find only a subset of the illegal states because its condition is only a weak version of the following characterization of terminal SCCs. Let *S* be a set of states such that from any state in *S*, there exists a sequence of transitions leading outside of *S*, and such that there are no transitions from states outside of *S* to any state in *S*. Then no state of *S* is part of a terminal SCC. (This characterization is exact since the set of all states not in terminal SCCs satisfies these conditions.) In Figure 28, the entire set of states *S* satisfies this criterion and is in fact illegal. Unfortunately, since all possible *S*'s cannot be feasibly checked, FILL may overestimate the set of legal states. In fact, it will not identify illegal states that form a cycle of length two or more, so in Figure 28, it will not find the illegal states *C* and *D*.

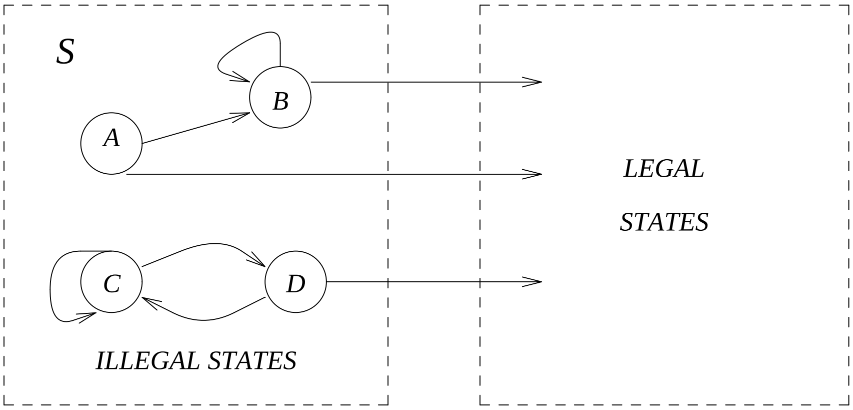


Figure 28. State-Space of a Sequential Machine

The reader is referred to [Long *et al.* 1995] for a detailed description of FILL.

6.2 Using Illegal States to Identify Sequentially Untestable Faults

Section 4.1 reviewed our previously proposed algorithm to identify redundant faults in a combinational circuit for which an illegal combination of values is a necessary condition for detection. The FIRE algorithm (Chapter 4) identifies combinational redundancies by using this procedure for processing illegal combinations consisting of conflicting assignments on every stem in the circuit. The FIRES algorithm (Chapter 5) extends FIRE to identify *c*-cycle redundancies. In this section, a new algorithm, FUNI, is presented, which identifies sequentially untestable faults for which the illegal states (of the fault-free circuit) computed by FILL are necessary for detection.

Illegal States and Sequential Implications

FUNI views an illegal state as an illegal combination of values and sequentially propagates uncontrollability and unobservability to identify sequentially untestable faults. The rules for propagating uncontrollability and unobservability in FUNI are similar to that in FIRES. However, when marking faults, FUNI does not perform the validation step performed in FIRES. Recall that FIRES performed the validation step to ensure that the implications leading to the conflict are also valid in certain time frames in the faulty circuit. This was required to prove that a fault which requires a conflict as a necessary condition for its detection is *c*-cycle redundant. FUNI does not need to perform a validation step because it only attempts to identify sequentially untestable as opposed to *c*-cycle redundant faults in FIRES. (An untestable fault identified by FUNI may not be redundant.) Moreover, FUNI analyzes illegal states as opposed to conflicts in FIRES.

Let $\{ l_1 = v_{1,}\ l_2 = v_{2,}\ \ldots,\ l_n = v_n \}$ be an illegal state, $v_i \in \{ 0,1 \}$. For every $l_a = v_a$, FUNI uses sequential uncontrollability and unobservability analysis to determine the set of faults $S^i_{v_a}$, which represents the set of faults whose activation in a time frame *k* or propagation to a PO from time frame *k*, requires $l_a = v_a$ (in the fault-free circuit) at *i* time units relative to *k*. Then the set of faults S^i that require $l_1 = v_1$, $l_2 = v_2$, $\cdots$, and $l_n = v_n$ at the same time is given by $S^i = S^i_{v_1} \cap S^i_{v_2} \cdots \cap S^i_{v_n}$. The intersections are done separately for the different time frames to ensure that the requirements to detect the fault really represent an illegal state.

Conceptually, the implications start propagating from the state variables involved in an illegal state at time 0 (see Figure 29). They propagate forward *f* time frames and backward *b* time frames. The time frames have positive or negative numbers depending on whether they are forward or backward time frames with respect to the starting time frame. This propagation is allowed for a maximum of T_M time frames ($f + b + 1 \leq T_M$). Figure 30 outlines the FUNI algorithm.

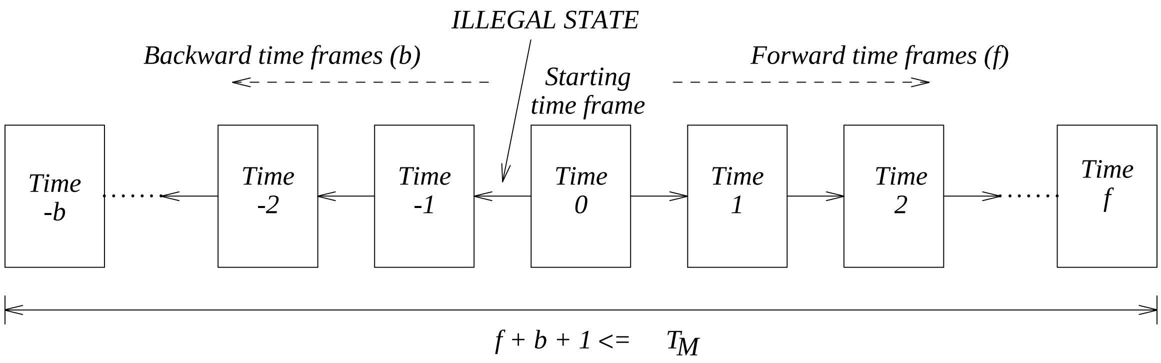


Figure 29. Model for Sequential Implications in FUNI

FUNI(T_M) {

/* T_M = Maximum number of time frames */

Compute illegal states (FILL)

For every illegal state, $I = \{ l_1 = v_1, l_2 = v_2, \cdots, l_n = v_n \}$ **{**

For every $l_a = v_a$ in I **{**

Sequentially Imply $l_a = \overline{v_a}$ over a maximum of T_M time frames. Let $S^i_{v_a}$ be the corresponding sets of uncontrollable and unobservable faults in time frame i.

}

Determine the set of faults, S_I, for which I is necessary for detection as $S_I = \bigcup_i \{S^i = S^i_{v_1} \cap S^i_{v_2}, \ldots, \cap S^i_{v_n}\}$.

}

The **untestable faults** are in the set $S_U = \bigcup_I S_I$.

}

Figure 30. Outline of the FUNI Algorithm

Theorem 2: A fault f identified by FUNI is untestable.

Proof: See Appendix B. □

Example 22: In Figure 31, $\{f = 1, g = 0\}$ is an illegal state. Table 15 summarizes the sequential implications resulting from $f = \overline{1}$ and $g = \overline{0}$. (A *s-a-v* fault on a line *l* is shown as l_v, and we use a collapsed set of faults based on fault equivalence.) The faults are maintained separately for each time frame. By processing a single illegal state, FUNI identifies three untestable faults; $\{h_0, c_1, d_0\}$. □

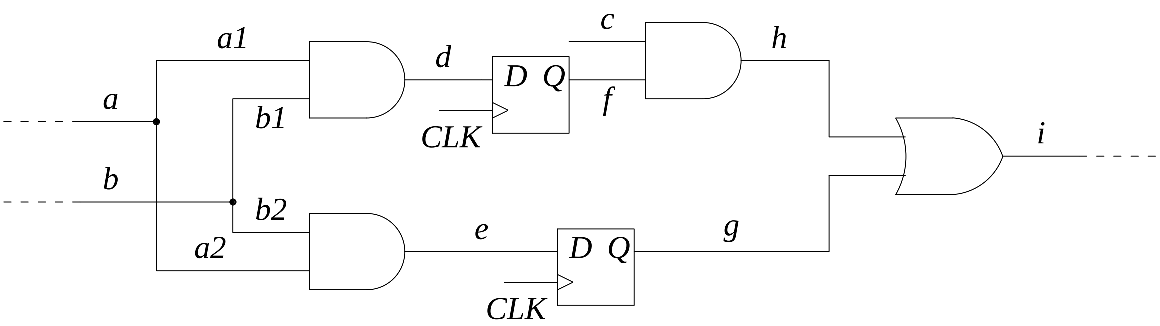


Figure 31. Circuit for Example 22

Table 15. Sequential Implications for Example 22

Process	**Time**	**Uncontrollability**	**Unobsservability**	**Collapsed Faults**
$f = \overline{1}$	0	$f = h = \overline{1}$	c	$S_f^0 = \{h_0, c_1\}$
	-1	$d = \overline{1}$	-	$S_f^{-1} = \{d_0\}$
	0	$g = i = \overline{0}$	h, f, c	$S_g^0 = \{i_1, h_0, f_1, c_1\}$
$g = \overline{0}$	-1	$e = b2 = a2 = = b =$ $a = b1 = a1 = d = \overline{0}$	$d, a1, b1$	$S_g^{-1} = \{e_1, b2_1, a2_1, b_1,$ $a_1, b1_1, a1_1, d_1, d_0\}$
	0	$f = \overline{0}$	-	$S_g^0 = \{i_1, h_0, f_1, c_1\}$
Untestable Faults	0	$S_f^0 \cap S_g^0 = \{h_0, c_1\}$		
	-1	$S_f^{-1} \cap S_g^{-1} = \{d_0\}$		

6.3 Experimental Results

FUNI was also implemented in C++ using the same object-oriented framework that was used to implement FIRES. FUNI was used to identify untestable faults in the ISCAS89 sequential benchmark circuits [Brglez *et al.* 1989]. Faults identified as untestable by FUNI were passed as the only targets for a conventional state-of-the-art

Table 16. FUNI Results for Benchmark Circuits

Circuit Name	FUNI					GENTEST			Speed-up Ratio
	# Fr.	# Unt.	CPU sec. F	CPU sec. U	CPU sec. T	# Unt.	# Abo.	CPU sec.	
S208	6	52	0.1	0.1	0.2	52	0	9.7	49
S298	4	21	0.4	0.3	0.7	16	5	269.8	385
S344	2	5	2.7	3.1	5.8	3	2	1491.1	257
S349	2	7	2.7	3.0	5.7	4	3	404.7	71
S382	3	13	1.3	0.5	1.8	6	7	1428.4	794
S386	2	36	0.1	0.1	0.2	22	14	781.5	3908
S400	3	16	1.2	0.5	1.7	5	11	1766.4	1039
S420	14	206	0.2	1.0	1.2	206	0	2085.5	1738
S444	3	23	3.9	0.5	4.4	5	18	1621.6	369
S510	2	0	0.2	0.1	0.3	-	-	-	-
S526	2	62	14.0	0.9	14.9	3	59	2094.1	141
S526n	2	65	15.4	0.8	16.2	4	61	2394.7	148
S641	15	55	1.0	0.5	1.5	55	0	0.0	0
S713	15	76	1.0	0.5	1.5	76	0	0.0	0
S820	2	26	0.2	0.2	0.4	11	15	851.9	2130
S832	2	28	0.3	0.2	0.5	11	17	1087.9	2176
S838	15	519	0.6	6.5	7.1	10	509	1684.3	237
S953	5	0	3.6	2.4	6.0	-	-	-	-
S1196	3	0	1.8	0.2	2.0	-	-	-	-
S1238	3	0	1.8	0.3	2.1	-	-	-	-
S1423	2	0	11.7	0.1	11.8	-	-	-	-
S1488	2	15	0.3	0.3	0.6	9	6	1660.0	2767
S1494	2	14	0.3	0.3	0.6	9	5	1545.7	2576
S5378	15	398	30.1	8.8	38.9	16	382	1412.9	36
S9234	15	93	71.7	23.0	94.7	81	12	1305.9	14
S13207	10	617	80.9	52.2	133.1	604	13	1297.8	10
S15850	10	629	63.9	32.5	96.4	612	17	1277.2	13
S35932	0	0	96.4	0.0	96.4	-	-	-	-
S38417	5	303	230.5	104.4	334.9	13	290	1372.9	4
S38584	5	671	219.7	45.9	265.6	453	218	2011.1	8

sequential test generator (GENTEST [Chakraborty *et al.* 1991]).

In Table 16, *#Fr.* is the number of time frames which FUNI processed, *#Unt.* is the number of identified untestable faults, and *#Abo.* is the number of targets aborted by GENTEST. *CPU times* are in seconds for a SUN sparc10. *F* is the time spent by FILL for building the BDDs for the FF logic, partitioning, identifying illegal states, and generating a prime cover of the illegal state set. *U* is the time to process the illegal states and to find the untestable faults. $T = F + U$ is the total CPU time used by FUNI. Speed-up ratio is (GENTEST time)/(FUNI time).

For example in S386, FUNI found 36 untestable faults in 0.2 seconds. GENTEST found only 22 of them to be untestable in 781 seconds and aborted 14 faults. Thus, in this case, FUNI is about 3908 times faster than GENTEST. For all circuits except four (S208, S420, S641 and S713), GENTEST could not solve all the faults identified by FUNI. In all circuits except two (S641 and S713), FUNI CPU times were much better than GENTEST. In many cases, FUNI was over *3 orders of magnitude faster* than GENTEST. On the average, FUNI was about 29 times faster than GENTEST and only 58% of the faults identified as untestable by FUNI were proved untestable by GENTEST.

Table 17. Comparison of [FUNI+FIRE] With [SFT+TRAN]

Circuit	FUNI		FIRE		[FUNI+FIRE]		[SFT+TRAN]	
Name	# Unt.	CPU secs.	# Red.	CPU secs.	# Unt.	CPU secs.	# Unt.	CPU secs.
S208	52	0.2	0	0.2	52	0.4	60	6.3
S298	21	0.7	0	0.2	21	0.9	35	30.6
S344	5	5.8	0	0.2	5	6.0	7	14.2
S349	7	5.7	2	0.1	7	5.8	9	14.4
S382	13	1.8	0	0.2	13	2.0	4	75.3
S386	36	0.2	0	0.2	36	0.4	70	75.0
S400	16	1.7	0	0.3	16	2.0	10	76.6
S420	206	1.2	0	0.5	206	1.7	221	15.3
S444	23	4.4	11	0.3	24	4.7	18	92.6
S510	0	0.3	0	0.6	0	0.9	0	4.8
S526	62	14.9	0	0.3	62	15.2	27	769.6
S526n	65	16.2	0	0.3	65	16.5	26	609.2
S641	55	1.5	0	0.2	55	1.7	0	123.2
S713	76	1.5	32	0.2	91	1.7	38	128.2
S820	26	0.4	0	0.3	26	0.7	35	1798.3
S832	28	0.5	0	0.3	28	0.8	51	1910.2
S838	519	7.1	0	1.5	519	8.6	549	87.4
S953	0	6.0	0	2.5	0	8.5	10	209.7
S1196	0	2.0	0	1.5	0	3.5	0	57.3
S1238	0	2.1	6	2.0	6	4.1	69	66.4
S1423	0	11.8	5	0.5	5	12.3	14	999.8
S1488	15	0.6	0	1.2	15	1.8	40	21781.9
S1494	14	0.6	1	1.2	15	1.8	51	20911.3
S5378	398	38.9	34	4.6	414	43.5	750	9703.1
S9234	93	94.7	165	14.1	257	108.8	524	2793.6
S13207	617	133.1	55	17.7	654	150.8	961	4329.7
S15850	629	96.4	234	17.6	816	114.0	448	4197.5
S35932	0	96.4	3984	141.0	3984	237.4	3984	486.4
S38417	303	334.9	99	38.5	381	373.4	391	6995.5
S38584	671	265.6	1037	139.8	1582	405.4	2142	9076.3

Table 17 also presents a comparison with the untestable faults that were identified by using the single-fault theorem (SFT) [Agrawal and Chakradhar 1993] and the TRAN combinational test generator [Chakradhar *et al.* 1993]. These are unpublished results and were recently obtained [Chakradhar 1994].

Since the results of [SFT + TRAN] include *all* the combinational redundancies in these circuits, we compare it with [FUNI + FIRE]. We show the number of redundant faults identified by FIRE alone. The [FUNI + FIRE] columns represent the union of the faults found by both algorithms and the total CPU time for a SUN sparc10. The [SFT + TRAN] experiments were performed on a SUN sparc2 (1 SUN sparc10 second = approximately 2 SUN sparc2 seconds) with a maximum of 8 time frames for the smaller benchmarks and 3 time frames for the larger benchmarks (S9234 and up). The [SFT + TRAN] results include circuit compilation time, unlike [FUNI + FIRE] results. In general, [SFT + TRAN] is likely to find more untestable faults at the expense of more CPU time. For S5378, [FUNI + FIRE] found 414 faults in 78 seconds (39 sparc10 seconds = approximately 78 sparc2 seconds), whereas [SFT + TRAN] found 750 faults in 9703 seconds. However, even though FUNI finds untestable faults without search, the following example shows that it can find a fault that can never be found by [SFT + TRAN] even with exhaustive search.

Example 23: The combinational ATPG theorems [Agrawal and Chakradhar 1993] will never find the fault *a s-a*-0 in the circuit of Figure 32 as untestable. FUNI identifies this fault as untestable; its detection requires the illegal state $\{ f = 1, h = 0 \}$ (for fault propagation). □

Consider, for example, circuit S38417. [FUNI + FIRE] found 732 untestable faults in 747 (sparc2) seconds, whereas [SFT + TRAN] found only 391 faults in about 7000 seconds.

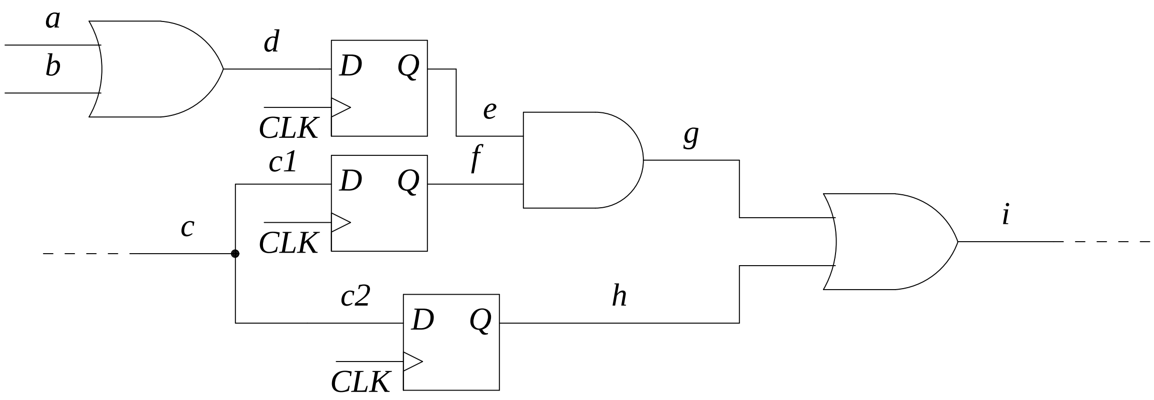


Figure 32. Circuit for Example 23

CHAPTER VII

RELATIONS BETWEEN FAULTS IDENTIFIED BY DIFFERENT ALGORITHMS

Chapter 4 presented the FIRE algorithm to identify combinational redundancies and Chapter 5 presented the FIRES algorithm, which extended the concept of FIRE to identify *c*-cycle redundancies. Chapter 6 presented another algorithm, FUNI, to find untestable faults using illegal states of the fault-free circuit. All these algorithms are fault-independent, do not use the search process encountered in ATG-based approaches, and use similar implication mechanisms. However, this does not mean that the faults identified by one algorithm will also be identified by the others. In this chapter, we analyze the relations between the sets of faults identified by FIRE, FIRES and FUNI. Whereas all faults identified as redundant by FIRE will also be identified as 0-cycle redundant by FIRES, it will be shown that, in general, FIRES and FUNI identify different (but partially overlapping) sets of faults. In this chapter, a sequentially redundant fault refers to a purely sequentially redundant fault that is combinationally testable.

7.1 Combinational and Sequential Redundancies

Faults that are untestable in a combinational circuit or in the full-scan version of a sequential circuit are combinationally redundant. However, a combinationally testable fault may be sequentially redundant.

FIRES incrementally performs implications over time frames and identifies *c*-cycle redundancies. For the starting time frame, FIRES also performs the same implications that FIRE would perform. Hence faults identified by FIRE will definitely be a subset of those identified by FIRES. However, as illustrated by the following example, FIRES may identify combinationally redundant faults that escape FIRE.

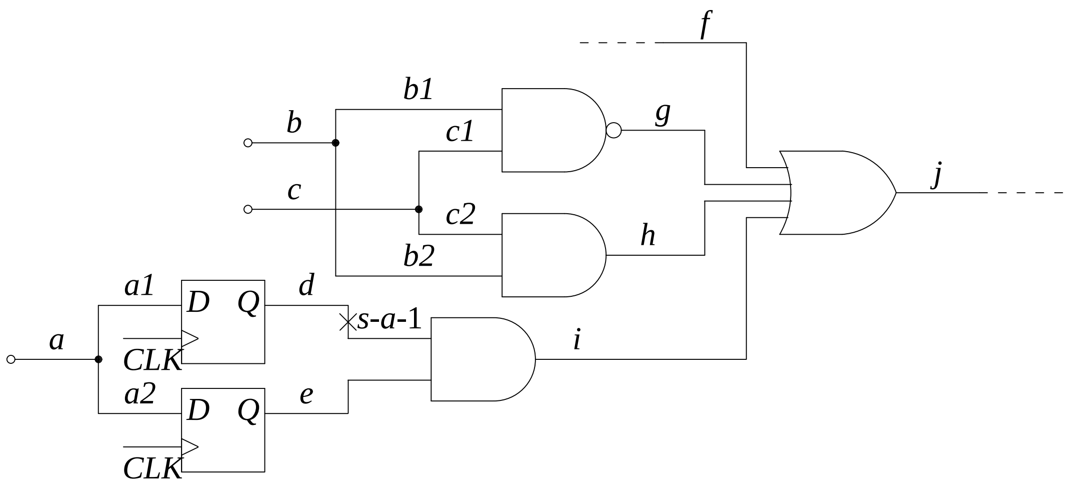


Figure 33. Combinationally Redundant Fault Identified By FIRES and Not By FIRE

Example 24: Consider the circuit in Figure 33 and the fault d *s-a*-1. When combinational ATG is performed for this fault, it is redundant because, the propagation requirement of $g = h = 0$ leads to a conflict on either stem b or stem c. But, this combinationally redundant fault will not be identified by FIRE because neither conflict is a necessary condition for the detection of this fault. However, the requirements $d = 0$ and $e = 1$ to detect d *s-a*-1 lead to a conflict on stem a in the previous time frame. Therefore, FIRES will identify it as a 1-cycle redundant fault. □

The above example also shows that FIRES may identify a combinationally redundant fault as a c-cycle redundant fault, $c > 0$. This happens because the fault has multiple causes for sequential untestability. Thus, a limitation of FIRES is that, in general, it cannot distinguish between combinationally and sequentially redundant faults. However, if the implications in FIRES do not propagate through FFs, then the faults it identifies will be combinationally redundant.

7.2 Faults Identified by FUNI

This subsection presents several examples to illustrate that untestable faults identified by FUNI may be combinationally redundant and may or may not be identified by FIRE and FIRES.

The following example illustrates that there could be combinationally redundant faults identified by FUNI and FIRE.

Example 25: Consider the circuit in Figure 34 and the fault *c s-a-*1. It can be verified that FIRE will identify this fault as combinationally redundant, because the propagation of its effect through gates *f* and *g* requires a conflict on stem *b*. Also, assuming that FILL will identify the state $\{c, d\} = \{0, 1\}$ as illegal, FUNI will identify this fault as untestable. □

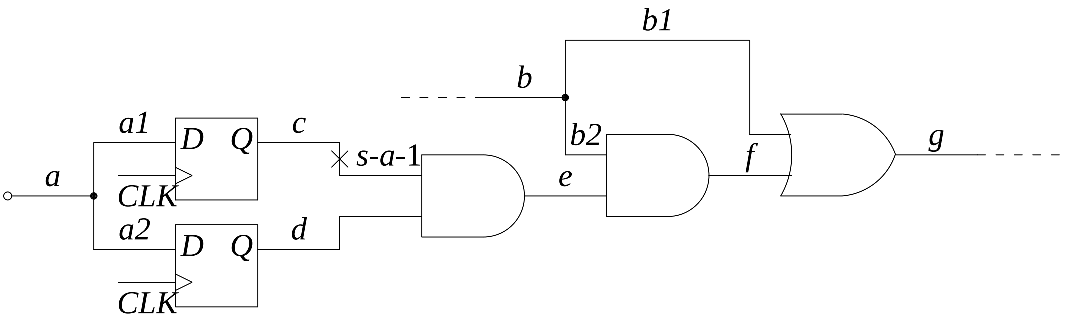


Figure 34. Combinationally Redundant Fault Identified By FIRE and FUNI

While it is obvious that FIRE can identify a combinationally redundant fault that will not be identified by FUNI, the above example illustrates that FUNI can identify a combinationally redundant fault (that is also identified by FIRE). Thus, FUNI cannot distinguish between combinationally redundant and sequentially untestable faults because of the existence of multiple causes for the untestability of some faults.

Since faults identified by FIRE form a subset of those identified by FIRES, based on the previous examples, we can also conclude that

- There could be faults identified by FIRES and not by FUNI.
- There could be faults identified by FIRE, FIRES and FUNI.

The following example illustrates that there could be faults identified by FIRES and not by FIRE and FUNI.

Example 26: Consider the circuit in Figure 35 and the fault *a s-a*-1. The fault is combinationally testable and hence will not be identified by FIRE. The only illegal state in the circuit is $\{e, d\} = \{1, 0\}$. However, this illegal state of the fault-free circuit is not necessary for the detection of *a s-a*-1 and hence FUNI will not identify it. However, the requirements for the sequential detection of this fault leads to a conflict on stem *b*. It can be verified that FIRES identifies this fault as a 0-cycle redundant fault, by processing a conflict on stem *b*. □

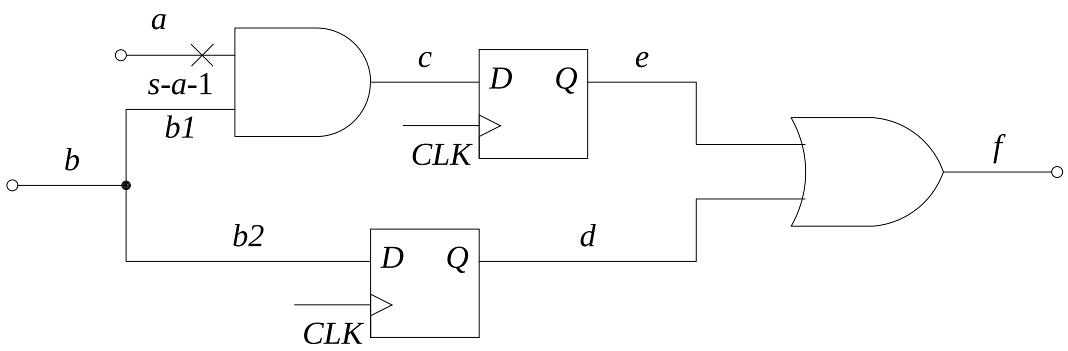


Figure 35. Fault Identified by FIRES and Not By FUNI and FIRE

Lastly, we illustrate that there could be untestable faults identified by FUNI and not by FIRES.

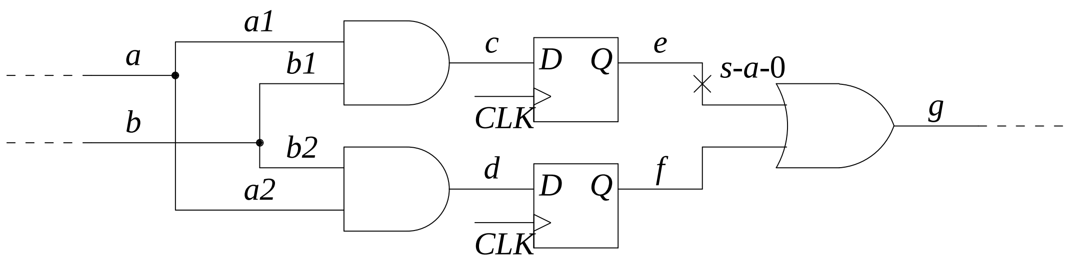


Figure 36. Untestable Fault Identified By FUNI and Not By FIRES

Example 27: Any test to detect the fault *e s-a*-0 in Figure 36 will require the fault-free circuit to be driven into the illegal state $\{e, f\} = \{1, 0\}$. Assuming, that FILL will identify this state as illegal, FUNI will identify this fault as untestable. However, note that a conflict on either stem a or stem b is not necessary for the detection of this fault. Consequently, FIRES will not identify it as redundant. □

The above example also illustrates that the requirement of an illegal state to detect a fault can lead to conflicts on a subset of stems in the circuit. However, this is not always the case. Untestability can occur because of illegal states without the presence of any conflicts. The following example illustrates this phenomenon.

Example 28: Consider the circuit in Figure 37. The circuit can be initialized by an input vector $a = 0$. After initialization, the circuit locks itself into state 0 and the output $c2$ is always 0. Consider the fault a *s*-*a*-1. The fault is an untestable and irredundant FPI because the faulty circuit is uninitializable. By processing the illegal state $c = 1$, FUNI will identify this fault as untestable. Note that when a sequential test generator attempts to derive a test sequence for this fault, its justification procedure will start looping over the unjustified state $c = 1$. Consequently, it will conclude that the fault is untestable. Note that the circuit does not contain any reconvergent fanout and hence during test generation no conflict can occur. □

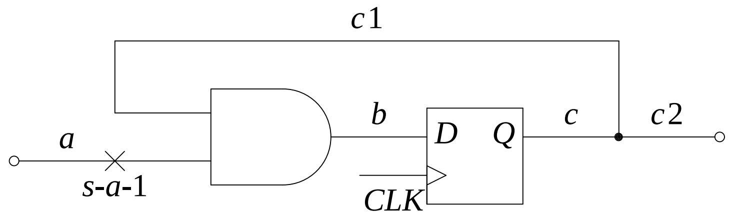


Figure 37. Circuit for Example 28

7.3 Summary

The following is a summary of the various examples presented in this chapter to illustrate the relations between faults identified by FIRE, FIRES and FUNI. Figure 38 pictorially illustrates all these cases.

1. Faults identified by FIRE form a subset of those identified by FIRES.
2. Combinationally redundant faults that are not identified by FIRE may be identified by FIRES as c-cycle redundant faults, $c > 0$.
3. FIRES can identify c-cycle redundant faults ($c \geq 0$) that are not identified by FIRE, but will be identified as untestable by FUNI (region 1 in the figure.)
4. There could be redundant faults identified by FIRES and not by FIRE and FUNI (region 2 in the figure.)
5. There could be (combinationally redundant) untestable faults that are identified by FUNI and FIRE (region 3 in the figure.)
6. There could be combinationally redundant faults that are identified by FIRE and not by FUNI (region 4 in the figure.)
7. There could be untestable faults identified by FUNI and not by FIRES (region 5 in the figure.)

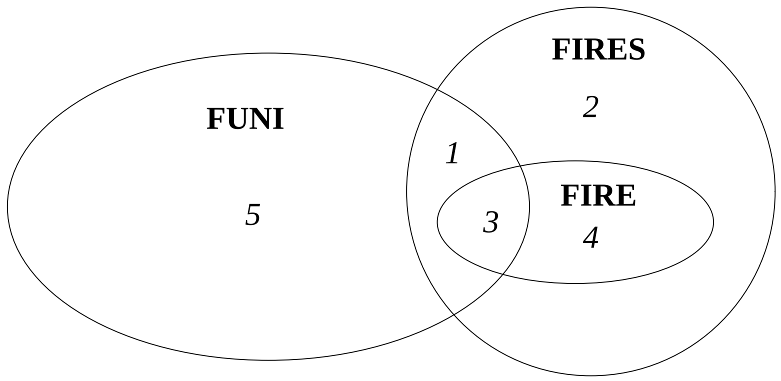


Figure 38. Relations Between Faults Identified By FIRE, FIRES, and FUNI

A common property of all the faults identified by FIRES and FUNI is that they are all untestable. Hence, an efficient preprocessor to a test generator can use the combined strengths of FIRES and FUNI to quickly prune some untestable faults. The effectiveness of this approach is presented in the following section.

7.4 Experimental Results

In this subsection, the effectiveness of using FIRES and FUNI in conjunction is demonstrated. The union of the faults identified by FIRES and FUNI were passed as the only targets to a conventional state-of-the-art sequential test generator (GENTEST [Chakraborty *et al.* 1991]).

Table 18. Comparison of [FIRES+FUNI] With GENTEST

Circuit	FIRE		FIRES		FUNI		[FIRES+FUNI]		GENTEST			Speed-up
Name	# Red.	CPU secs.	# Unt.	CPU secs.	# Unt.	CPU secs.	# Unt.	CPU secs.	# Unt.	# Abo.	CPU secs.	ratio
S208	0	0.2	60	0.9	52	0.2	60	1.1	60	0	12.7	12
S298	0	0.2	0	0.3	21	0.7	21	1.0	16	5	269.8	270
S344	0	0.2	0	0.3	5	5.8	5	6.1	3	2	1491.1	244
S349	2	0.1	2	0.3	7	5.7	7	6.0	4	3	404.7	68
S382	0	0.2	0	0.9	13	1.8	13	2.7	6	7	1428.4	529
S386	0	0.2	27	0.5	36	0.2	38	0.7	23	15	738.4	1055
S400	0	0.3	1	0.9	16	1.7	16	2.6	5	11	1766.4	679
S420	0	0.5	217	5.7	206	1.2	217	6.9	217	0	2216.2	321
S444	11	0.3	11	1.3	23	4.4	24	5.7	6	18	1861.7	327
S510	0	0.6	0	0.8	0	0.3	0	1.1	-	-	-	-
S526	0	0.3	0	0.7	62	14.9	62	15.6	3	59	2094.1	134
S526n	0	0.3	0	0.7	65	16.2	65	16.9	4	61	2394.7	142
S641	0	0.2	0	0.5	55	1.5	55	2.0	55	0	0.0	0
S713	32	0.2	32	0.6	76	1.5	91	2.1	79	12	770.9	367
S820	0	0.3	0	0.4	26	0.4	26	0.8	11	15	851.9	1065
S832	0	0.3	0	0.4	28	0.5	28	0.9	11	17	1087.9	1209
S838	0	1.5	529	25.4	519	7.1	529	32.5	7	522	1714.4	53
S953	0	2.5	0	4.5	0	6.0	0	10.5	-	-	-	-
S1196	0	1.5	0	2.2	0	2.0	0	4.2	-	-	-	-
S1238	6	2.0	6	2.5	0	2.1	6	4.6	6	0	0.8	0
S1423	5	0.5	5	1.2	0	11.8	5	13.0	2	3	890.1	69
S1488	0	1.2	0	1.5	15	0.6	15	2.1	9	6	1660.0	791
S1494	1	1.2	1	1.5	14	0.6	15	2.1	10	5	1549.9	738
S5378	34	4.6	273	37.3	398	38.9	457	76.2	16	441	1532.2	20
S9234	165	14.1	284	90.2	93	94.7	350	184.9	328	22	1469.5	8
S13207	55	17.7	661	77.0	617	133.1	707	210.1	678	29	1155.7	6
S15850	234	17.6	332	312.4	629	96.4	805	408.8	785	20	1311.0	3
S35932	3984	141.0	3984	525.6	0	96.4	3984	622.0	0	3984	6397.0	10
S38417	99	38.5	147	295.5	303	334.9	381	630.4	14	367	1742.8	3
S38584	1037	139.8	1397	234.4	671	265.6	1626	500.0	1214	412	2802.9	6

Table 18 shows the results. For convenience, the faults identified individually by FIRE, FIRES and FUNI and their respective CPU times are also shown. All CPU times are for a SUN sparc10. The results of FIRES are those without the validation step in Table 11 (Chapter 5). In almost all cases, [FIRES+FUNI] performed much better than

GENTEST.

For example, in S386, no combinational redundancies were identified by FIRE, 27 untestable faults were identified by FIRES (without validation) and 36 untestable faults were identified by FUNI. The number of faults in the union of the faults identified by FIRES and FUNI is 38 and the total CPU time is 0.7 seconds. GENTEST could only prove untestability for 23 of these 38 faults. The remaining 15 were aborted. Also, GENTEST took 1055 times more CPU time to achieve this result.

On the average, GENTEST aborted 63% of the [FIRES+FUNI] faults and took 14 times more CPU time.

CHAPTER VIII

CONCLUSIONS

While combinational ATG is a problem where current state-of-the-art algorithms can efficiently deal with VLSI circuits, the same cannot be said about the much more complex sequential ATG problem, which may require extremely long run-times. Typically, a sequential test generator spends most of its time dealing with untestable faults. Untestable faults are faults that do not have any tests. Redundant faults are faults whose presence does not change the input-output behavior of the circuit. In sequential circuits, redundant faults form a subset of the untestable faults. In spite of their inertness, there are several reasons why they create problems. First, from a synthesis viewpoint, the circuit can be simplified by removing the regions associated with redundant faults. Second, we can eliminate redundant and untestable faults from the target fault list for a test generation process and save on testing effort. This is because these faults do not have any tests and the test generator exhibits its worst-case behavior when dealing with them. However, none of this would be possible unless we have some way of identifying untestable and redundant faults in a circuit, without using test generation search effort. This thesis presents efficient algorithms to identify redundancy and untestability in synchronous sequential circuits, without relying on test generation search effort.

The initial part of this thesis reveals many surprises encountered when trying to solve the sequential RID problem. Since sequentially untestable faults are often mislabeled as redundant faults, examples were provided to illustrate the differences between the two. Existing solutions for sequential RID were analyzed and it was shown that some of these solutions are based on incorrect theoretical results. Specifically, it was shown that untestable faults in balanced pipeline circuits, untestable faults in circuits with fault-free global reset mechanisms, and unactivatable faults are not necessarily redundant. Also it was shown that analyzing the mandatory assignments that lead to conflicts do not necessarily identify redundancy. It was pointed out that adding a (fault-free) global reset mechanism to a circuit or retiming synchronous circuitry may create redundancies. The limitations of some other methods were also presented.

This thesis has presented FIRES, a new fault-independent redundancy identification algorithm for sequential circuits. The concept of a *c*-cycle redundant fault is introduced, where the behavior of the fault-free circuit is indistinguishable from that of the faulty circuit which has been clocked for at least c times. The conventional notion of redundancy is a special case of a *c*-cycle redundancy with $c = 0$. An important result of this thesis is that a fault whose detection requires a conflict is *c*-cycle redundant. FIRES also determines the value of c. If a circuit is simplified by removing a *c*-cycle redundancy, the new circuit need only be clocked with c arbitrary input vectors before using the original initialization sequence. No assumption about a global reset mechanism is made and no state transition graph information is needed. FIRES processes conflicts on stems and finds faults for which these conflicts are necessary for detection.

Any practical state-of-the-art sequential test generator cannot distinguish between untestable and redundant faults. FIRES identifies redundancies in provably *worst-case polynomial time complexity*. FIRES provides the *first efficient solution to the sequential RID problem, practical for large circuits*. The redundancies found by FIRES are not restricted to any particular classes of faults, nor is the algorithm restricted for small circuits. FIRES, however, finds only a subset of the redundant faults. This is because there could be redundant faults that cannot be detected due to conflicts on multiple stems. Moreover, the implications performed are not guaranteed to be complete and pessimistic heuristics are used to decide when multiple-path sensitization is precluded. The use of improved implication procedures [Chakradhar *et al.* 1993, Kunz and Pradhan

1992] is likely to increase the number of redundancies that can be identified using FIRES at the cost of additional CPU time.

Logic synthesis procedures for redundancy removal are very time-consuming if they rely on ATG for RID. FIRES can be easily integrated with any synthesis system to remove redundancies. Apart from being able to identify sequential redundancy in a practical manner, FIRES has significant advantages in redundancy removal applications because it is fault-independent and the average CPU time it takes to process the entire circuit is negligible. After a redundancy is removed, new redundancies may be created in the circuit. Even combinational redundancy removal using ATG is expensive because many faults may have to be retargeted over multiple passes of test generation to identify newly created redundancies. However, FIRES may at most have to reanalyze previously analyzed stems in such an iterative procedure. This operation is far cheaper than retargeting many faults for test generation. Hence FIRES is expected to perform better than any ATG-based approach for redundancy removal.

This thesis also introduced FUNI, an algorithm to identify untestable faults using illegal states. FUNI uses the illegal states of the fault-free circuit computed by FILL [Long *et al.* 1995] and identifies faults for which these illegal states are necessary for detection. The results presented show that the untestable faults found by FUNI are not easy targets for a conventional state-of-the-art sequential test generator. In fact, FUNI was *several orders of magnitude* faster than a conventional ATG algorithm that targeted only the faults identified by FUNI. Moreover, about 42% of these faults (found by FUNI without search) are aborted by the ATG algorithm using exhaustive search.

A deterministic sequential test generator performing exhaustive search can prove untestability for the faults found by FIRES and FUNI (without search), if given enough time. Both FIRES and FUNI are fault-independent, and by processing a single conflict or illegal state, they may simultaneously identify several untestable faults that would require separate targeting by an ATG-based approach. In general, FIRES and FUNI identify different (but partially overlapping) sets of untestable faults and these faults are not easy targets for state-of-the-art practical sequential test generators. Moreover, the [FIRES+FUNI] CPU times are negligible compared to the complete sequential test generation runs for all faults in the circuit. Thus, [FIRES+FUNI] can be used as a preprocessing step to any state-of-the-art sequential test generator to obtain significant savings in computation time and to increase the detectable fault coverage. It is currently being integrated as a preprocessor to GENTEST [Chakraborty *et al.* 1991]

8.1 Future Work

Though FIRES provides an efficient solution to identify c-cycle redundancies, a few more problems remain unsolved with respect to redundancy removal. Every redundant fault identified by FIRES has a c associated with it. Removal of the region for a c-cycle redundant fault requires that the new circuit be clocked at least c times before use. However, determination of the minimum c for the new circuit after a sequence of c-cycle redundancy removals merits further investigation. To accomplish this, relationships between the c-cycle redundant faults that were removed to generate the new circuit will have to be understood. Another problem is the creation of new redundancies after a redundancy is removed. A redundancy removal procedure using FIRES will have to reanalyze the stems that were already analyzed before the current redundancy removal. Alternatively, this procedure may be used in conjunction with the fault-independent procedure to identify newly created redundancies after a redundancy removal, that was proposed in [Abramovici and Iyer 1992, Iyer 1991]. Redundancy removal using FIRES is currently under investigation.

The entire concept of processing possible conflicts and illegal states of the the fault-free circuit and finding faults that require them as necessary conditions for detection has important applications in delay testing. It could be used to identify untestable delay faults without any search. The conditions for detecting a delay fault are, in general, more stringent than those for detecting stuck-at faults. Thus the chances for conflicts are higher in delay testing and usually, a large fraction of the modeled faults are untestable. Thus extension of the FIRES algorithm to identify untestable delay faults would be very useful. Moreover, it would be the first fault-independent algorithm to identify untestable delay faults.

The faults identified by FIRES and FUNI are untestable under the multiple observation time test strategy. With respect to test generation, this has significant advantages because these faults will remain untestable no matter how sophisticated the test generator is. DFT techniques attempt either to make the test generation simpler or to decrease the number of untestable faults. Several DFT techniques exist in the literature. FIRES and FUNI could thus serve as good starting points for DFT techniques that aim at increasing the number of testable faults in the circuit with minimal area and performance overheads.

APPENDIX A

PROOF OF *c*-CYCLE REDUNDANCY FOR FIRES

In this section, it is shown that a fault *f* found by FIRES is c_f-cycle redundant. We begin with a lemma which proves that to demonstrate *c*-cycle redundancy, it is enough to show that the following restricted scenario does not occur.

Lemma 2: Assume the fault *f* is not *c*-cycle redundant. Then there exists an initial state S_p for the faulty circuit, a state S_c that is reachable from S_p after applying *c* input vectors, and an input sequence *I*, such that if we start *both* fault-free and faulty circuits from S_c and apply *I*, then:

1. a fault effect from *f* is generated in the first time frame; and
2. there exists a sensitized path from this fault effect to a PO.

Proof: Since the fault *f* is not *c*-cycle redundant, it is testable from some state S_c' of the faulty circuit that can be reached after power-up and the application of some sequence of *c* input vectors. Let S_p' be the power-up state. Since the fault is testable from S_c', there exists an input sequence I' that will distinguish the faulty circuit started in state S_c' from the fault-free circuit started from any state. In particular, if the fault-free circuit is also started in state S_c', then I' must eventually activate the fault and propagate the effect to a PO.

Consider the first time frame where a fault effect is observed at a PO. There must exist a sensitized path from the PO (in this time frame) back to at least one fault site where the fault is activated. Among all such sites, consider the one which occurs earliest, and let the time frame for this first activation be F_a. Define the state of the faulty circuit at this point as S_c. Note that since S_c is reachable from S_c' and since S_c' can be reached from S_p' in *c* cycles, there must be a state which can reach S_c in *c* cycles; choose S_p to be such a state. Define *I* as the subsequence of vectors of I' applied after the faulty circuit reaches S_c.

Now we have defined the S_p, S_c, and *I* stated in the lemma's conclusion. The only difficulty is that as the faulty circuit is driven from S_c' to S_c, the fault-free circuit may be driven from S_c' to some state S_g *different from* S_c (because of fault activations during this time). At the moment, we only know that if we start the fault-free circuit from S_g and the faulty circuit from S_c and apply I, then a difference will be observed at the PO chosen above. We now claim that if we change the starting state of the fault-free circuit to S_c, a fault effect will still be observed. We know that with the circuits in states S_c and S_g, the fault is being activated and there exists a sensitized path to the PO. If changing the state of the fault-free circuit to S_c does not affect this activation and sensitization, we are done. Otherwise, some previous fault effects (reflected in the differences between S_g and S_c) must be reaching either the activation site or a side input on the sensitized path. By eliminating these previous fault effects, we are destroying either the activation or the sensitization. But this means that there must be a time frame prior to F_a where the fault is being activated and for which a sensitized path exists from the fault site to the PO. However F_a was chosen to be the first time frame satisfying these conditions, a contradiction. Hence this case cannot occur, and so by changing the fault-free circuit's starting state from S_g to S_c, we still observe a fault effect at the PO. □

Proof of Theorem 1

Assume that f is not c_f-cycle redundant. (Without loss of generality, we may assume that f is a stuck-at-1 fault, say on line m.) Then the conditions in the conclusion of Lemma 2 hold. This scenario is illustrated in Figure 39. Let k be the length of I. We denote the initial activation time frame by F_a, and we say that F_a is one of the ‘‘machine’’ time frames.

Let s be the stem that FIRES analyzed to find f. Let zero be the starting time frame for FIRES where uncontrollability began propagating from the stem. FIRES identified f

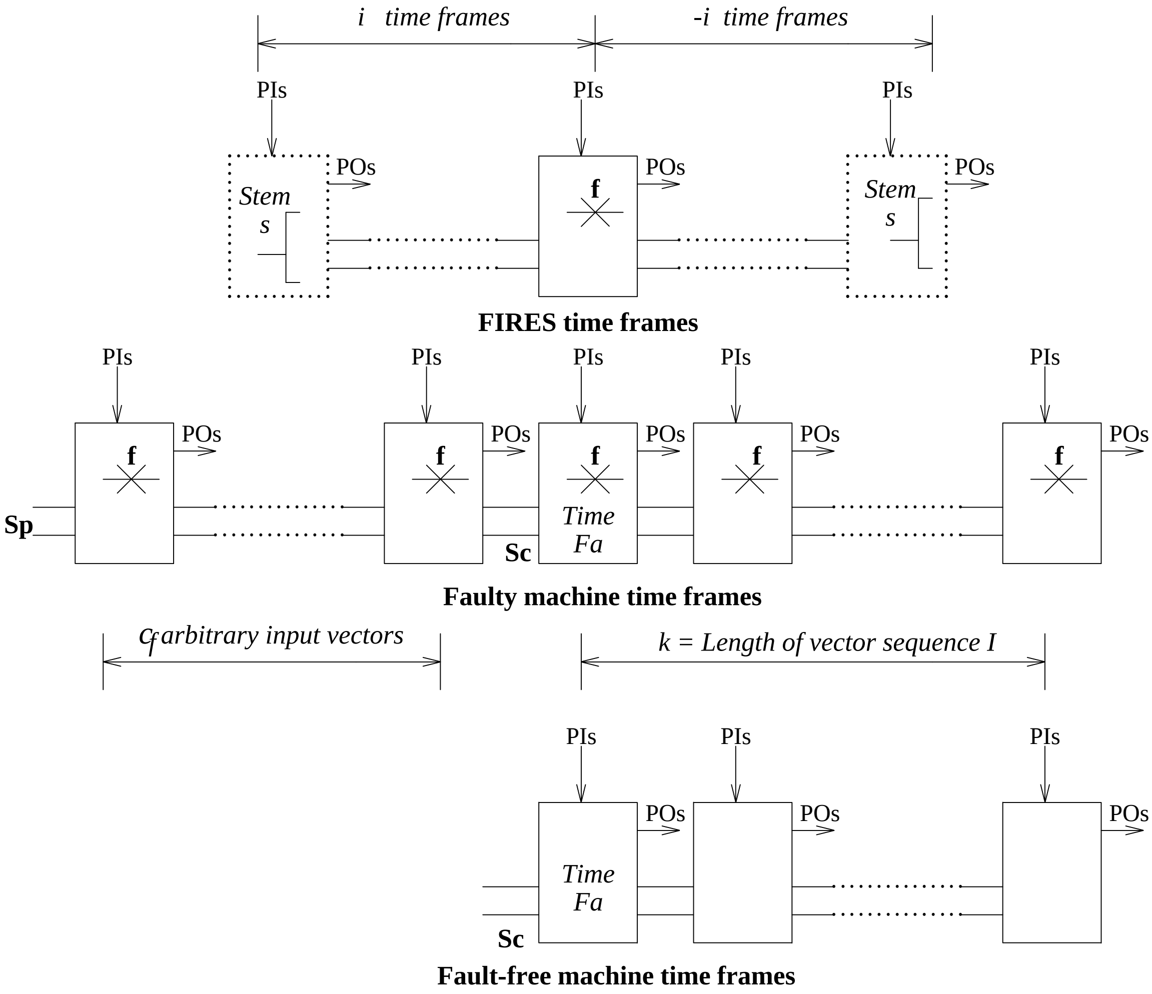


Figure 39. Figure Used in Proof of Theorem 1

in some time frame *i*, where *i* can be negative, positive or zero. We align the time frame *i* of FIRES with F_a, as illustrated in Figure 39. (Note that the stem *s* may be ahead of or behind time frame *i*; the two cases are combined in the figure.) This means that to translate FIRES time frame numbers to machine time frame numbers, we add $F_a - i$. Let *b* be the number of backwards time frames needed by FIRES to find the fault. By the rules for computing c_f, we know that $-b + (F_a - i)$ is at least $F_a - c_f$. This tells us that leftmost time frame used by FIRES (when expressed as a machine time frame) cannot be before S_p. Also, if the rightmost time frame used by FIRES is beyond the rightmost machine time frame, then we can append (arbitrary) additional vectors to I to extend the machine frames. (This does not affect activation of the fault and the propagation of its effect to a PO.) Thus, we ensure that all of the time frames used by FIRES have corresponding machine time frames. From now on, we will switch back and forth between FIRES time frame numbers and machine time frame numbers implicitly. For example, if we say ''the value of a line in frame i,'' it should be understood to refer to FIRES time frame i (since it is in the context of FIRES where i was defined) and machine time frame $i + (F_a - i)$.

Since FIRES identified the fault, it must have done so both when the stem s was set to $\overline{0}$ and when it was set to $\overline{1}$. In order to argue that no fault effect makes it to a PO, thus obtaining a contradiction, we will use the implications done by FIRES during one of these experiments. To determine which set of implications to use, we examine the value on the stem s in the machine time frames. If FIRES time frame zero is (strictly) to the left of F_a, then let u be the value on stem s in the faulty circuit (in that frame); otherwise, let u be the value on s in the fault-free circuit. Without loss of generality, we may assume that u is 0. We then consider the analysis of FIRES when it set s to $\overline{1}$.

Note that the implications from a $\overline{1}$ on s and the implications from a 0 on the same stem would be completely analogous if we were only dealing with the fault-free circuit. That is, if FIRES put a $\overline{0}$ on some line l in time frame j ($l^j = \overline{0}$), we could conclude that the fault-free circuit had $l^j = 1$, and similarly if FIRES had $l^j = \overline{1}$, the fault-free circuit would have $l^j = 0$. We would like to use this analogy between the way values propagate to conclude that in time frame F_a and to its right, whenever l^j has value $\overline{0}$ in some FIRES time frame, then it has value 1 in the corresponding fault-free machine time frame, and similarly for $\overline{1}$ and 0. The only problem with doing this is that machine time frames to the left of F_a do not exist for the fault-free circuit, and during the analysis by FIRES, values may have propagated back and forth across the left boundary of frame F_a. The state S_c may not even be reachable from any other state in the fault-free circuit, so there is no hope in general of extending the fault-free machine time frames to the left. Thus, we must make do with implications in the faulty circuit. *The validation step in FIRES exactly checks that any implications done in time frames to the left of F_a are valid in the faulty circuit.* Also note that when an implication crosses the boundary from F_a into the previous frame or from the previous frame into F_a, we can transfer values between the faulty and fault-free circuits because S_c (and the input) is the same in both circuits. These facts allow us to conclude that every implication done by FIRES in finding *f* is either represented by a corresponding value in the faulty circuit for a frame to the left of F_a, or by a corresponding value in the fault-free circuit in time frame F_a or its successors.

Now let us consider how FIRES found *f*. Recall that we are assuming that *f* is the fault *m s-a*-1. Thus, FIRES may have discovered *f* by implying that the value on *m* (in F_a) was a $\overline{0}$. By the above reasoning, in the fault-free circuit the value on *m* in this frame must be 1, and since the logic driving *m* is before the fault, this logic must be producing a 1 in the faulty circuit too. Thus the fault is not being activated in frame F_a, a contradiction. The other possibility is that FIRES marked the fault because of an unobservability indicator on *m*. By the rules for propagation of unobservability, there must exist a set $\{p^j\}$ of lines (in time frames F_a and beyond) such that:

1. every path from *m* (in F_a) to a PO passes through a gate that has some line $l^j \in \{p^j\}$ as a side input;
2. the value on l^j corresponds to a controlling value for that gate; and
3. there is no path from a copy of *m* in any time frame between F_a and *j* (inclusive) to l^j.

By the second point and the reasoning above, we know that l^j in the fault-free circuit has a controlling value. The third point tells us that no fault effect can propagate onto l^j, so this line also has a controlling value in the faulty circuit. Finally, by the first point, we see that every path from *m* (in F_a) to a PO passes through some gate that has a side input with a controlling value in both circuits. Hence the fault effect from m cannot make it to a PO, a contradiction. Thus, I in fact does not activate the fault in the first time frame and propagate the effect to a PO, and hence f is c_f-cycle redundant. □

APPENDIX B

PROOF OF UNTESTABILITY FOR FUNI

Let f (l s-a-v) be a fault identified by FUNI after processing an illegal state I. Let i be an integer to represent the relative distance between the starting time frame (the time frame where the uncontrollability indicators corresponding to I started propagating) and the time frame where FUNI found the fault. Also assume that there exists a test to detect f. Let L represent all the legal states of the fault-free circuit. Any state in L must have at least one legal predecessor state and all legal descendant states. Since the fault is detectable, it can be shown in a manner similar to the proof of Lemma 2 that there exists a state $S_L \in L$ and an input sequence I_{S_L} such that starting both fault-free and faulty circuits from S_L and applying I_{S_L} will immediately activate the fault and propagate the effect to a PO at some time.

For each component of I, $i_n = v_n$, FUNI must have found fault f as either uncontrollable or unobservable at time i relative to the starting time frame. Let I_U be the components of I, processing which FUNI found f as uncontrollable at time i; i.e any vector which drives value $\bar{v}$ on line l at some time k in the fault-free circuit must realize *every component of* I_U at i time units relative to k. If i is negative, then after starting the fault-free circuit in state S_L and applying I_1 and $|i|$ clocks, the states that would be reached would be covered by I_U. If i is positive, then all predecessors of S_L which require i clock cycles to reach S_L must be covered by I_U. If $i = 0$, then S_L must be covered by I_U. This is illustrated in Figure 40. All cases of i are covered in the same figure, by showing the starting time frame with dashed lines. Note that for any particular scenario only one of these dashed time frames will exist.

After activating the fault, I_{S_L} must propagate the fault-effect to some PO at some time o. In doing so, the fault-effect from the first activation time frame may combine with fault-effects from subsequent time frames. Let I_O be the components of I, processing which FUNI found f as unobservable at time i relative to the starting time frame. For every component of I_O, by the rules of propagation of unobservability, there

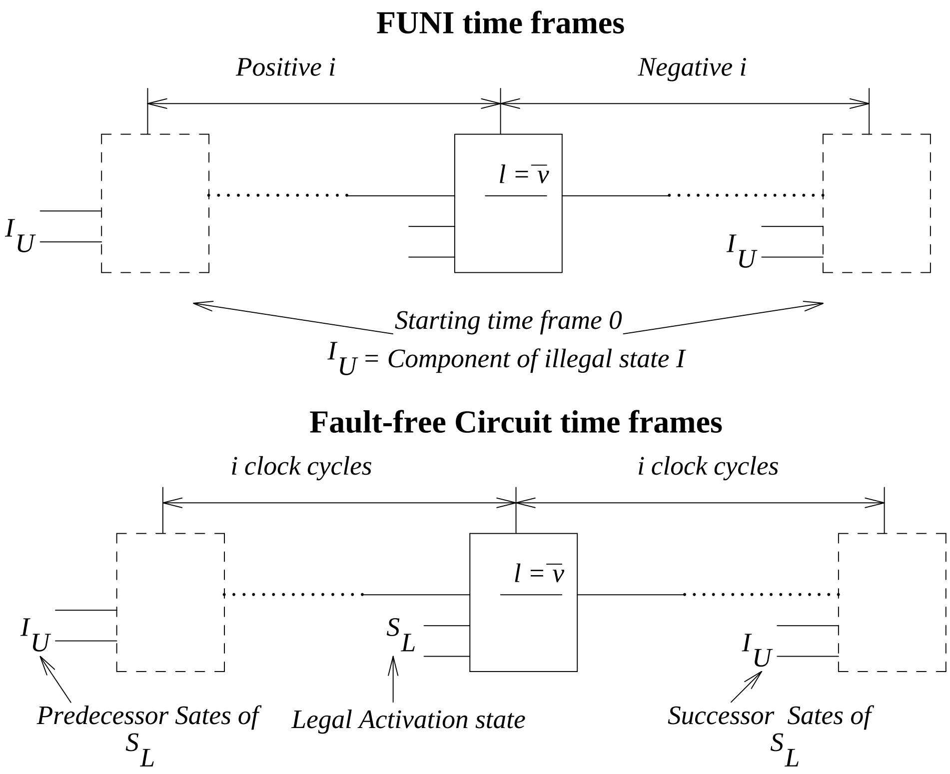


Figure 40. Uncontrollability Propagations in FUNI

must exist a set $\{p^j\}$ of lines (in time frames between time i and time o) such that:

1. every path from l (in time i) to a PO passes through a gate at some time k, $i \leq k \leq o$ that has some line $l^j \in \{p^j\}$ as a side input;

2. l^j has an uncontrollability indicator at time k contributing to the unobservability on l at time i.

3. there is no path from a copy of l in any time frame between i and k (inclusive) to l^j.

By condition 1, I_{S_L} must propagate the fault-effect through at least one gate G having a side input as $l_s \in \{p^j\}$. Since the fault-effect is propagating through G, by conditions 2 and 3, I_{S_L} must set l_s to the non-controlling value of G in both fault-free and faulty circuits. Note that condition 3 ensures that a fault-effect does not propagate onto l_s; i.e. the non-controlling value of G will be required in both fault-free and faulty circuits in order to propagate a fault-effect through G. Figure 41 illustrates a scenario for unobservability propagations in FUNI for negative i.

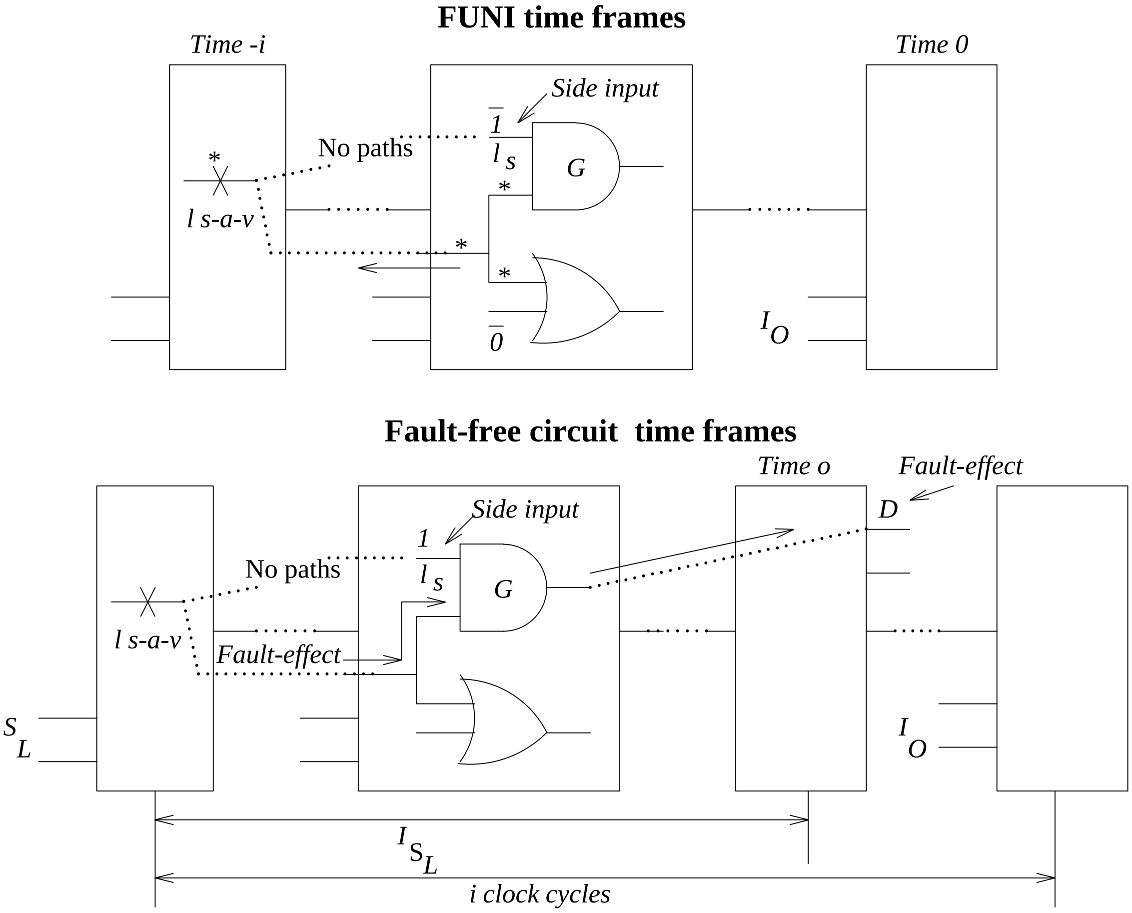


Figure 41. Unobservability Propagations in FUNI

Here, by condition 2, the fault-free circuit must realize I_O at i time units relative to the time frame where the fault was first activated. If i is negative, it means that after starting the fault-free circuit in state S_L and applying $|i|$vectors the states that would be reached must be covered by I_O. If i is positive, then all predecessors of S_L which require i clock cycles must be covered by I_O. If $i = 0$ then S_L must be covered by I_O.

Now, $I = I_U + I_O$ and i is the same for each component of I that FUNI processed. Since I is illegal, any state covered by I is also illegal. We now show that for all possible values of i our initial assumption that there exists a test is contradicted.

- If i is negative, then starting the fault-free circuit in state S_L, there exists a sequence of length $|i|$ such that it can reach a state covered by I. This is impossible because S_L is a legal state.
- If i is positive, then all predecessors of S_L that require i clock cycles to reach S_L must be covered by I. This is impossible because for any legal state S_L, there exists at least one legal predecessor state that can reach S_L in i clock cycles.
- If $i = 0$, then S_L must be covered by I. This is impossible because $S_L \in L$.

Thus, for all initial-state pairs that require both circuits to start from from the same legal state of the fault-free circuit, it can be shown there exists no test to detect f. Hence f is sequentially untestable. □

REFERENCES

[Abramovici and Breuer 1979] M. Abramovici and M. A. Breuer, "On Redundancy and Fault Detection in Sequential Circuits." *IEEE Trans. on Computers*, vol. C-28, no. 11, pp. 864-865, November 1979.

[Abramovici *et al.* 1990] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design.* Computer Science Press, New York, NY, 1990.

[Abramovici and Iyer 1992] M. Abramovici and M. A. Iyer, "One-Pass Redundancy Identification and Removal." *Proc. Intn'l. Test Conf.*, pp. 807-815, September 1992.

[Abramovici *et al.* 1986] M. Abramovici, J. J. Kulikowski, P. R. Menon, and D. T. Miller, "SMART and FAST: Test Generation for VLSI scan-design circuits." *IEEE Design and Test of Computers*, pp. 43-54, August 1986.

[Abramovici *et al.* 1991] M. Abramovici, J. J. Kulikowski, and R. K. Roy, "The Best Flip-Flops to Scan," *Proc. Intn'l Test Conf.*, pp. 166-173, October 1991.

[Abramovici *et al.* 1992] M. Abramovici, D. T. Miller, and R. K. Roy, "Dynamic Redundancy Identification in Automatic Test Generation." *IEEE Trans. on CAD*, pp. 404-407, March 1992.

[Abramovici and Parikh 1992] M. Abramovici and P. S. Parikh, "Warning: 100% Fault Coverage May Be Misleading!!." *Proc. Intn'l. Test Conf.*, pp. 662-668, September 1992.

[Agrawal and Chakradhar 1993] V. D. Agrawal and S. T. Chakradhar, "Combinational ATPG Theorems for Identifying Untestable Faults in Sequential Circuits." *Proc. European Test Conf.*, pp. 249-253, April 1993.

[Breuer 1978] M. A. Breuer, "New Concepts in Automated Testing of Digital Circuits," *Proc. EEC Symp. on CAD of Digital Electronic Circuits and Systems*, pp. 69-92, North-Holland Publishing Co., Nov. 1978.

[Brglez *et al.* 1989] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits", *Proc. 1989 Intn'l. Symposium on Circuits and Systems*, pp. 1929-1934, May 1989.

[Brglez and Fujiwara 1985] F. Brglez and H. Fujiwara, "Neutral netlist of ten combinational benchmark circuits and a target translator in FORTRAN," *Proc. IEEE Intn'l. Symposium on Circuits and Systems*, June 1985.

[Bryan *et al.* 1989] D. Bryan, F. Brglez, and R. Lisanke, "Redundancy Identification and Removal." *Proc. MCNC Workshop on Logic Synthesis*, 1989.

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

[Chakraborty *et al.* 1991] T. J. Chakraborty, S. Davidson, and B. Bencivenga, "GENTEST: The Architecture of Sequential Circuit Test Generator," *Proc. Custom Integrated Circuits Conf.*, May 1991.

[Chakradhar 1994] S. T. Chakradhar, *Personal Communication*, October 1994.

[Chakradhar *et al.* 1993] S. T. Chakradhar, V. D. Agrawal and S. G. Rothweiler, "A Transitive Closure Algorithm for Test Generation." *IEEE Trans. on CAD*, vol. 12, no. 7, pp. 1015-1028, July 1993.

[Chakradhar *et al.* 1995] S. T. Chakradhar, S. G. Rothweiler, and V. D. Agrawal, "Redundancy Removal and Test Generation for Circuits with Non-Boolean Primitives." *13th. IEEE VLSI Test Symp.*, pp. 12-19, May 1995.

[Cheng 1991a] K. T. Cheng, "On Removing Redundancy in Sequential Circuits." *Proc. 28th. ACM/IEEE Design Automation Conf.*, pp. 164-169, June 1991.

[Cheng 1991b] K. T. Cheng, "An ATPG-Based Approach to Sequential Logic Optimization." *Proc. Intn'l. Conf. on Computer Aided Design*, pp. 372-375, November 1991.

[Cheng 1993] K. T. Cheng, "Redundancy Removal for Sequential Circuits Without Reset States," *IEEE Trans. on CAD*, vol. 12, no. 1, pp. 13-24, January 1993.

[Cho *et al.* 1993b] H. Cho, G. D. Hachtel, E. Macii, B. Plessier, and F. Somenzi, "Algorithms for Approximate FSM Traversal," *Proc. 30th. ACM/IEEE Design Automation Conf.*, pp. 25-30, June 1993.

[Cho *et al.* 1993c] H. Cho, G. D. Hachtel, and F. Somenzi, "Redundancy Identification/Removal and Test Generation for Sequential Circuits Using Implicit State Enumeration." *IEEE Trans. on CAD*, vol. 12, no. 7, pp. 935-945, July 1993.

[Cho *et al.* 1993a] H. Cho, S. W. Jeong, F. Somenzi, and C. Pixley "Synchronizing Sequences and Symbolic Traversal Techniques in Test Generation," *Journal Of Electronic Testing (JETTA): Theory and Applications*, vol. 4, pp. 19-31, February 1993

[Coudert *et al.* 1989] O. Coudert, C. Berthet, and J. C. Madre, "Verification of Synchronous Sequential Machines Based on Symbolic Execution," *Proc. of the 1989 Intn'l Workshop on Automatic Verification Methods for Finite State Systems*, Grenoble, France, vol. 407 of *Lecture Notes in Computer Science,* Springer-Verlag, June 1989.

[Coudert and Madre 1989] O. Coudert and J. C. Madre, "Symbolic Computation of the Valid States of a Sequential Machine: Algorithms and Discussion," *Proc. of the 1991 Intn'l Workshop on Formal Methods in VLSI Design*, January 1991.

[Davidson 1994] S. Davidson, "Is I_{DDQ} Yield Loss Inevitable ?" *Proc. Intn'l. Test Conf.*, pp. 572-579, Oct. 1994.

[Devadas *et al.* 1990] S. Devadas, H. K. T. Ma, and A. R. Newton, "Redundancies and Don't Cares in Sequential Logic Synthesis." *J. Electronic Testing: Theory and Application (JETTA)*, vol. 1-1, pp. 15-30, February 1990.

[Dey and Chakradhar 1994] S. Dey and S. T. Chakradhar, "Retiming Sequential Circuits to Enhance Testability," *12th. IEEE VLSI Test Symp.*, pp. 28-33, April 1994.

[El-Maleh *et al.* 1994] A. El-Maleh, T. E. Marchok, J. Rajski, and W. Maly, ‘‘Behavior and Testability Preservation Under the Retiming Transformation,’’ *McGill University Technical Report #94-R3*, December 1994.

[Entrena and Cheng 1993] L. Entrena and K. T. Cheng, "Sequential Logic Optimization By Redundancy Addition and Removal," *Proc. Intn’l. Conf. on CAD*, pp. 310-315, November. 1993.

[Friedman 1967] A. D. Friedman, " Fault Detection in Redundant Circuits." *IEEE Trans. on Electronic Computers*, vol. EC-16, pp. 99-100, February 1967.

[Giraldi and Bushnell 1991] J. Giraldi and M. L. Bushnell, "Search State Equivalence for Redundancy Identification and Test Generation." *Proc. Intn’l. Test Conf.*, pp. 184-193, October 1991.

[Harihara and Menon 1989] M. Harihara and P. R. Menon, "Identification of Undetectable Faults in Combinational Circuits." *Proc. Intn’l. Conf. on Computer Design*, pp. 290-293, October 1989.

[Hayes 1976] J. P. Hayes, "On the Properties of Irredundant Logic Networks." *IEEE Trans. on Computers*, vol. C-25, No. 9, pp. 884-892, September 1976.

[Ibarra and Sahni 1975] O. H. Ibarra and S. K. Sahni, "Polynomially Complete Fault Detection Problems." *IEEE Trans. on Computers*, vol. C-24, pp. 242-249, March 1975.

[Iyer 1991] M. A. Iyer, "One-Pass Redundancy Identification and Removal in Combinational Circuits." *MS Thesis, Dept. of Electrical and Computer Engineering, Illinois Institute of Technology, Chicago, IL*, December 1991.

[Iyer and Abramovici 1994a] M. A. Iyer and M. Abramovici, "Low-Cost Redundancy Identification for Combinational Circuits," *Proc. 7th. Intn’l. Conf. on VLSI Design, India*, pp. 315-318, January 1994.

[Iyer and Abramovici 1994b] M. A. Iyer and M. Abramovici, "FIRE: A Fault-Independent Redundancy Identification Algorithm for Combinational Circuits," *IEEE Trans. on VLSI Systems*, (first revision under review).

[Iyer and Abramovici 1994c] M. A. Iyer and M. Abramovici, "Sequentially Untestable Faults Identified Without Search (Simple Implications Beat Exhaustive Search!)," *Proc. Intn’l. Test Conf.*, pp. 259-266, October 1994.

[Iyer *et al.* 1995] M. A. Iyer, D. E. Long, and M. Abramovici, "Identifying Sequential Redundancies Without Search," To be published (hopefully, sometime in the near future).

[Iyer *et al.* 1996] M. A. Iyer, D. E. Long, and M. Abramovici, "Surprises in Sequential Redundancy Identification," *9th. Intn’l. Conf. on VLSI Design, India*, January 1996 (submitted).

[Kajihara *et al.* 1992] S. Kajihara, H. Shiba and K. Kinoshita, "Removal of Redundancy in Logic Circuits under Classification of Undetectable Faults." *Proc. 22nd Fault-Tolerant Computing Symposium*, pp. 263-270, July 1992.

[Kunz and Pradhan 1992] W. Kunz and D. K. Pradhan, "Recursive Learning: An Attractive Alternative to the Decision Tree for Test Generation in Digital Circuits." *Proc. Intn'l. Test Conf.*, pp. 816-825, September 1992.

[Leiserson and Saxe 1991] C. E. Leiserson and J. B. Saxe, "Retiming Synchronous Circuitry," *Algorithmica*, vol. 6, pp. 5-35, 1991.

[Liang *et al.* 1994] H.-C Liang, C. L. Lee, and J. E. Chen, "A Sequential Redundant Fault Identification Scheme and Its Application to Test Generation," *Proc. Asian Test Symp.*, pp. 57-62, 1994.

[Lin *et al.* 1990] B. Lin, H. J. Touati, and A. R. Newton, "Don't Care Minimization of Multi-Level Sequential Logic Networks," *Proc. Intn'l Conf. on Computer-Aided Design*, pp. 414-417, November 1990.

[Long *et al.* 1995] D. E. Long, M. A. Iyer and M. Abramovici, "Identifying Sequentially Untestable Faults Using Illegal States," *Proc. 13th. IEEE VLSI Test Symp.*, pp. 4-11, May 1995.

[Marchok *et al.* 1994] T. E. Marchok, A. El-Maleh, W. Maly, and J. Rajski, "Complexity of Sequential ATPG," *European Design and Test Conf.*, 1994.

[Menon and Ahuja 1992] P. R. Menon and H. Ahuja, "Redundancy Removal and Simplification of Combinational Circuits." *Digest of Papers, IEEE VLSI Test Symposium*, pp. 268-273, April 1992.

[Moondanos and Abraham 1992] J. Moondanos and J. A. Abraham, "Sequential Redundancy Identification Using Verification Techniques." *Proc. Intn'l. Test Conf.*, pp. 197-205, September 1992.

[Niermann and Patel 1991] T. Niermann and J. H. Patel, "HITEC: A Test Generation Package for Sequential Circuits," *Proc. European Conf. on Design Automation*, pp. 214-218, 1991.

[Pixley 1990] C. Pixley, "A Computational Theory and Implementation of Sequential Hardware Equivalence," *Computer-Aided Verification*, E. M. Clarke and R. P. Kurshan, editors, pp. 293-320, 1990.

[Pomeranz and Reddy 1992] I. Pomeranz and S. M. Reddy, "The Multiple Observation Time Test Strategy." *IEEE Trans. on Computers*, vol. 41, no. 5, pp. 627-637, May 1992.

[Pomeranz and Reddy 1993] I. Pomeranz and S. M. Reddy, "Classification of Faults in Synchronous Sequential Circuits," *IEEE Trans. on Computers*, vol. 42, pp. 1066-1077, September 1993.

[Pomeranz and Reddy 1994a] I. Pomeranz and S. M. Reddy, "On Identifying Undetectable and Redundant Faults in Synchronous Sequential Circuits," *12th. IEEE VLSI Test Symp.*, pp. 8-14, April 1994.

[Pomeranz and Reddy 1994b] I. Pomeranz and S. M. Reddy, "On Achieving Complete Testability of Synchronous Sequential Circuits with Synchronizing Sequences," *Proc. Intn'l. Test Conf.*, pp. 1007-1016, October 1994.

[Pomeranz *et al.* 1993] I. Pomeranz, S. M. Reddy and J. H. Patel, "Theory and Practice of Sequential Machine Testing and Testability." *Proc. 22nd. Fault-Tolerant Computing Symposium*, pp. 1-8, July 1992.

[Ratiu *et al.* 1982] I. M. Ratiu, A. Sangiovanni-Vincentelli, and D. O. Pederson, "VICTOR: a fast VLSI testability analysis program." *Proc. 1982 Intn'l. Test Conf.*, pp. 397-401, November 1982.

[Rutman 1972] R. A. Rutman, "Fault Detection Test Generation for Sequential Logic by Heuristic Tree Search," *IEEE Computer Group Repository,* Paper No. R-72-187, 1972.

[Singhal *et al.* 1995b] V. Singhal, C. Pixley, A. Aziz and R. K. Brayton, "Delaying Safeness for More Flexibility," *Unpublished Technical Report from the University of California at Berkeley*, January 17, 1995.

[Singhal *et al.* 1995a] V. Singhal, C. Pixley, R. L. Rudell and R. K. Brayton, "The Validity of Retiming Sequential Circuits," *Proc. Design Automation Conf.*, June. 1995 (to appear).

[Touati *et al.* 1990] H. Touati, H. Savoj, B. Lin, R. K. Brayton, and A. Sangiovanni-Vincentelli "Implicit State Enumeration of Finite State Machines Using BDD's," *Proc. Intn'l Conf. on Computer-Aided Design*, pp. 130-133, November 1990.