THE UNIVERSITY OF CALGARY

A NOVEL TEST GENERATION SYSTEM FOR SEQUENTIAL CIRCUITS

by

Bin Du

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE

DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

CALGARY, ALBERTA

JUNE, 1994

ISBN 0-315-99344-8

Name Bin Du

Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.

Electronics and Electrical Engineering — SUBJECT TERM

0 5 4 4 — SUBJECT CODE

U·M·I

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS

Architecture	0729
Art History	0377
Cinema	0900
Dance	0378
Fine Arts	0357
Information Science	0723
Journalism	0391
Library Science	0399
Mass Communications	0708
Music	0413
Speech Communication	0459
Theater	0465

EDUCATION

General	0515
Administration	0514
Adult and Continuing	0516
Agricultural	0517
Art	0273
Bilingual and Multicultural	0282
Business	0688
Community College	0275
Curriculum and Instruction	0727
Early Childhood	0518
Elementary	0524
Finance	0277
Guidance and Counseling	0519
Health	0680
Higher	0745
History of	0520
Home Economics	0278
Industrial	0521
Language and Literature	0279
Mathematics	0280
Music	0522
Philosophy of	0998
Physical	0523
Psychology	0525
Reading	0535
Religious	0527
Sciences	0714
Secondary	0533
Social Sciences	0534
Sociology of	0340
Special	0529
Teacher Training	0530
Technology	0710
Tests and Measurements	0288
Vocational	0747

LANGUAGE, LITERATURE AND LINGUISTICS

Language	
General	0679
Ancient	0289
Linguistics	0290
Modern	0291
Literature	
General	0401
Classical	0294
Comparative	0295
Medieval	0297
Modern	0298
African	0316
American	0591
Asian	0305
Canadian (English)	0352
Canadian (French)	0355
English	0593
Germanic	0311
Latin American	0312
Middle Eastern	0315
Romance	0313
Slavic and East European	0314

PHILOSOPHY, RELIGION AND THEOLOGY

Philosophy	0422
Religion	
General	0318
Biblical Studies	0321
Clergy	0319
History of	0320
Philosophy of	0322
Theology	0469

SOCIAL SCIENCES

American Studies	0323
Anthropology	
Archaeology	0324
Cultural	0326
Physical	0327
Business Administration	
General	0310
Accounting	0272
Banking	0770
Management	0454
Marketing	0338
Canadian Studies	0385
Economics	
General	0501
Agricultural	0503
Commerce-Business	0505
Finance	0508
History	0509
Labor	0510
Theory	0511
Folklore	0358
Geography	0366
Gerontology	0351
History	
General	0578
Ancient	0579
Medieval	0581
Modern	0582
Black	0328
African	0331
Asia, Australia and Oceania	0332
Canadian	0334
European	0335
Latin American	0336
Middle Eastern	0333
United States	0337
History of Science	0585
Law	0398
Political Science	
General	0615
International Law and Relations	0616
Public Administration	0617
Recreation	0814
Social Work	0452
Sociology	
General	0626
Criminology and Penology	0627
Demography	0938
Ethnic and Racial Studies	0631
Individual and Family Studies	0628
Industrial and Labor Relations	0629
Public and Social Welfare	0630
Social Structure and Development	0700
Theory and Methods	0344
Transportation	0709
Urban and Regional Planning	0999
Women's Studies	0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

Agriculture	
General	0473
Agronomy	0285
Animal Culture and Nutrition	0475
Animal Pathology	0476
Food Science and Technology	0359
Forestry and Wildlife	0478
Plant Culture	0479
Plant Pathology	0480
Plant Physiology	0817
Range Management	0777
Wood Technology	0746
Biology	
General	0306
Anatomy	0287
Biostatistics	0308
Botany	0309
Cell	0379
Ecology	0329
Entomology	0353
Genetics	0369
Limnology	0793
Microbiology	0410
Molecular	0307
Neuroscience	0317
Oceanography	0416
Physiology	0433
Radiation	0821
Veterinary Science	0778
Zoology	0472
Biophysics	
General	0786
Medical	0760

EARTH SCIENCES

Biogeochemistry	0425
Geochemistry	0996
Geodesy	0370
Geology	0372
Geophysics	0373
Hydrology	0388
Mineralogy	0411
Paleobotany	0345
Paleoecology	0426
Paleontology	0418
Paleozoology	0985
Palynology	0427
Physical Geography	0368
Physical Oceanography	0415

HEALTH AND ENVIRONMENTAL SCIENCES

Environmental Sciences	0768
Health Sciences	
General	0566
Audiology	0300
Chemotherapy	0992
Dentistry	0567
Education	0350
Hospital Management	0769
Human Development	0758
Immunology	0982
Medicine and Surgery	0564
Mental Health	0347
Nursing	0569
Nutrition	0570
Obstetrics and Gynecology	0380
Occupational Health and Therapy	0354
Ophthalmology	0381
Pathology	0571
Pharmacology	0419
Pharmacy	0572
Physical Therapy	0382
Public Health	0573
Radiology	0574
Recreation	0575
Speech Pathology	0460
Toxicology	0383
Home Economics	0386

PHYSICAL SCIENCES

Pure Sciences

Chemistry	
General	0485
Agricultural	0749
Analytical	0486
Biochemistry	0487
Inorganic	0488
Nuclear	0738
Organic	0490
Pharmaceutical	0491
Physical	0494
Polymer	0495
Radiation	0754
Mathematics	0405
Physics	
General	0605
Acoustics	0986
Astronomy and Astrophysics	0606
Atmospheric Science	0608
Atomic	0748
Electronics and Electricity	0607
Elementary Particles and High Energy	0798
Fluid and Plasma	0759
Molecular	0609
Nuclear	0610
Optics	0752
Radiation	0756
Solid State	0611
Statistics	0463

Applied Sciences

Applied Mechanics	0346
Computer Science	0984
Engineering	
General	0537
Aerospace	0538
Agricultural	0539
Automotive	0540
Biomedical	0541
Chemical	0542
Civil	0543
Electronics and Electrical	0544
Heat and Thermodynamics	0348
Hydraulic	0545
Industrial	0546
Marine	0547
Materials Science	0794
Mechanical	0548
Metallurgy	0743
Mining	0551
Nuclear	0552
Packaging	0549
Petroleum	0765
Sanitary and Municipal	0554
System Science	0790
Geotechnology	0428
Operations Research	0796
Plastics Technology	0795
Textile Technology	0994

PSYCHOLOGY

General	0621
Behavioral	0384
Clinical	0622
Developmental	0620
Experimental	0623
Industrial	0624
Personality	0625
Physiological	0989
Psychobiology	0349
Psychometrics	0632
Social	0451

Nom ____

Dissertation Abstracts International est organisé en catégories de sujets. Veuillez s.v.p. choisir le sujet qui décrit le mieux votre thèse et inscrivez le code numérique approprié dans l'espace réservé ci-dessous.

____ ☐☐☐☐ U·M·I

SUJET | CODE DE SUJET

Catégories par sujets

HUMANITÉS ET SCIENCES SOCIALES

COMMUNICATIONS ET LES ARTS

Architecture	0729
Beaux-arts	0357
Bibliothéconomie	0399
Cinéma	0900
Communication verbale	0459
Communications	0708
Danse	0378
Histoire de l'art	0377
Journalisme	0391
Musique	0413
Sciences de l'information	0723
Théâtre	0465

ÉDUCATION

Généralités	515
Administration	0514
Art	0273
Collèges communautaires	0275
Commerce	0688
Économie domestique	0278
Éducation permanente	0516
Éducation préscolaire	0518
Éducation sanitaire	0680
Enseignement agricole	0517
Enseignement bilingue et multiculturel	0282
Enseignement industriel	0521
Enseignement primaire.	0524
Enseignement professionnel	0747
Enseignement religieux	0527
Enseignement secondaire	0533
Enseignement spécial	0529
Enseignement supérieur	0745
Évaluation	0288
Finances	0277
Formation des enseignants	0530
Histoire de l'éducation	0520
Langues et littérature	0279
Lecture	0535
Mathématiques	0280
Musique	0522
Orientation et consultation	0519
Philosophie de l'éducation	0998
Physique	0523
Programmes d'études et enseignement	0727
Psychologie	0525
Sciences	0714
Sciences sociales	0534
Sociologie de l'éducation	0340
Technologie	0710

LANGUE, LITTÉRATURE ET LINGUISTIQUE

Langues	
Généralités	0679
Anciennes	0289
Linguistique	0290
Modernes	0291
Littérature	
Généralités	0401
Anciennes	0294
Comparée	0295
Mediévale	0297
Moderne	0298
Africaine	0316
Américaine	0591
Anglaise	0593
Asiatique	0305
Canadienne (Anglaise)	0352
Canadienne (Française)	0355
Germanique	0311
Latino-américaine	0312
Moyen-orientale	0315
Romane	0313
Slave et est-européenne	0314

PHILOSOPHIE, RELIGION ET THÉOLOGIE

Philosophie	0422
Religion	
Généralités	0318
Clergé	0319
Études bibliques	0321
Histoire des religions	0320
Philosophie de la religion	0322
Théologie	0469

SCIENCES SOCIALES

Anthropologie	
Archéologie	0324
Culturelle	0326
Physique	0327
Droit	0398
Économie	
Généralités	0501
Commerce-Affaires	0505
Économie agricole	0503
Économie du travail	0510
Finances	0508
Histoire	0509
Théorie	0511
Études américaines	0323
Études canadiennes	0385
Études féministes	0453
Folklore	0358
Géographie	0366
Gérontologie	0351
Gestion des affaires	
Généralités	0310
Administration	0454
Banques	0770
Comptabilité	0272
Marketing	0338
Histoire	
Histoire générale	0578
Ancienne	0579
Médiévale	0581
Moderne	0582
Histoire des noirs	0328
Africaine	0331
Canadienne	0334
États-Unis	0337
Européenne	0335
Moyen-orientale	0333
Latino-américaine	0336
Asie, Australie et Océanie	0332
Histoire des sciences	0585
Loisirs	0814
Planification urbaine et régionale	0999
Science politique	
Généralités	0615
Administration publique	0617
Droit et relations internationales	0616
Sociologie	
Généralités	0626
Aide et bien-àtre social	0630
Criminologie et établissements pénitentiaires	0627
Démographie	0938
Études de l' individu et de la famille	0628
Études des relations interethniques et des relations raciales	0631
Structure et développement social	0700
Théorie et méthodes.	0344
Travail et relations industrielles	0629
Transports	0709
Travail social	0452

SCIENCES ET INGÉNIERIE

SCIENCES BIOLOGIQUES

Agriculture	
Généralités	0473
Agronomie.	0285
Alimentation et technologie alimentaire	0359
Culture	0479
Élevage et alimentation	0475
Exploitation des péturages	0777
Pathologie animale	0476
Pathologie végétale	0480
Physiologie végétale	0817
Sylviculture et faune	0478
Technologie du bois	0746
Biologie	
Généralités	0306
Anatomie	0287
Biologie (Statistiques)	0308
Biologie moléculaire	0307
Botanique	0309
Cellule	0379
Écologie	0329
Entomologie	0353
Génétique	0369
Limnologie	0793
Microbiologie	0410
Neurologie	0317
Océanographie	0416
Physiologie	0433
Radiation	0821
Science vétérinaire	0778
Zoologie	0472
Biophysique	
Généralités	0786
Medicale	0760

SCIENCES DE LA TERRE

Biogéochimie	0425
Géochimie	0996
Géodésie	0370
Géographie physique	0368
Géologie	0372
Géophysique	0373
Hydrologie	0388
Minéralogie	0411
Océanographie physique	0415
Paléobotanique	0345
Paléoécologie	0426
Paléontologie	0418
Paléozoologie	0985
Palynologie	0427

SCIENCES DE LA SANTÉ ET DE L'ENVIRONNEMENT

Économie domestique	0386
Sciences de l'environnement	0768
Sciences de la santé	
Généralités	0566
Administration des hipitaux	0769
Alimentation et nutrition	0570
Audiologie	0300
Chimiothérapie	0992
Dentisterie	0567
Développement humain	0758
Enseignement	0350
Immunologie	0982
Loisirs	0575
Médecine du travail et thérapie	0354
Médecine et chirurgie	0564
Obstétrique et gynécologie	0380
Ophtalmologie	0381
Orthophonie	0460
Pathologie	0571
Pharmacie	0572
Pharmacologie	0419
Physiothérapie	0382
Radiologie	0574
Santé mentale	0347
Santé publique	0573
Soins infirmiers	0569
Toxicologie	0383

SCIENCES PHYSIQUES

Sciences Pures

Chimie	
Genéralités	0485
Biochimie	487
Chimie agricole	0749
Chimie analytique	0486
Chimie minérale	0488
Chimie nucléaire	0738
Chimie organique	0490
Chimie pharmaceutique	0491
Physique	0494
PolymÇres	0495
Radiation	0754
Mathématiques	0405
Physique	
Généralités	0605
Acoustique	0986
Astronomie et astrophysique	0606
Electronique et électricité	0607
Fluides et plasma	0759
Météorologie	0608
Optique	0752
Particules (Physique nucléaire)	0798
Physique atomique	0748
Physique de l'état solide	0611
Physique moléculaire	0609
Physique nucléaire	0610
Radiation	0756
Statistiques	0463

Sciences Appliqués Et Technologie

Informatique	0984
Ingénierie	
Généralités	0537
Agricole	0539
Automobile	0540
Biomédicale	0541
Chaleur et ther modynamique	0348
Conditionnement (Emballage)	0549
Génie aérospatial	0538
Génie chimique	0542
Génie civil	0543
Génie électronique et électrique	0544
Génie industriel	0546
Génie mécanique	0548
Génie nucléaire	0552
Ingénierie des systämes	0790
Mécanique navale	0547
Métallurgie	0743
Science des matériaux	0794
Technique du pétrole	0765
Technique minière	0551
Techniques sanitaires et municipales	0554
Technologie hydraulique	0545
Mécanique appliquée	0346
Géotechnologie	0428
Matières plastiques (Technologie)	0795
Recherche opérationnelle	0796
Textiles et tissus (Technologie)	0794

PSYCHOLOGIE

Généralités	0621
Personnalité	0625
Psychobiologie	0349
Psychologie clinique	0622
Psychologie du comportement	0384
Psychologie du développement	0620
Psychologie expérimentale	0623
Psychologie industrielle	0624
Psychologie physiologique	0989
Psychologie sociale	0451
Psychométrie	0632

THE UNIVERSITY OF CALGARY

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled, "A NOVEL TEST GENERATION SYSTEM FOR SEQUENTIAL CIRCUITS", submitted by Bin Du in partial fulfillment of the requirements for the degree of Master of Science.

[illegible]

Dr. Jun Gu, Superviser & Chairman
Dept. of Electrical & Computer Engineering

[illegible]

Dr. E. P. Nowicki
Dept. of Electrical & Computer Engineering

[illegible]

Dr. Paul Kwok
Department of Computer Science

Date: Aug. 5, 1994

ABSTRACT

This thesis presents a new approach for test generation of sequential circuits. The problem of test generation for sequential circuits is decomposed into three subproblems, i.e., excitation vector generation, state justification, and state differentiation. By disabling all flip-flops in a sequential circuit, the sequential circuit is transformed into a pseudo-combinational circuit. Then an extended transitive closure algorithm extracts the implication graph and the SAT formula from the model of the circuit incorporating necessary conditions for fault activation and path sensitization. To enhance the efficiency of state differentiation in the existing three-phase ATPG, a novel backward deterministic method for state differentiation is proposed. The new test generation algorithm has been tested using the ISCAS'89 benchmarks. The algorithm yielded a high fault coverage and is shown to be very efficient in generating tests for large size sequential circuits. The experimental results on large sequential circuits indicate that, our approach is much faster than the existing deterministic test generation algorithms.

Acknowledgement

I do not know how to express my sincere thanks to my supervisor, Jun Gu. It is my greatest fortune to have Jun as my advisor. I appreciate so much for Jun's insight, inspiration, and constant encouragement throughout my graduate research work. He is not only the best advisor I ever have, he is also the best friend for me and my family. The friendship between us is what has been keeping me going for these years, and it will keep me going for the rest of my life. Jun's deep insight into the difficult problems is always a precious resource for me. Without him, my stay at Calgary would not have been meaningful. I have been convinced that Jun is a dedicated educator and first class researcher, a role model which I found very difficult to follow. He has provided me tremendous support whenever I run into a difficult situation. There is no way for me to express my gratitude to him in any words.

I am grateful to Ruchir Puri who gave me many constructive discussion during this research work. I would also like to thank Abdel Yousif who was kind enough to read the contents of this thesis during its preparation and provided me with helpful comments. I thank many friends at the University of Calgary, for those enjoyable moments which they have been so generous to share with me. I would also like to gratefully acknowledge the financial support provided by NSERC and the Department of Electrical and Computer Engineering at the University of Calgary.

Finally and most of all, I am grateful to my wife, Xiaoying. As a husband who is engaged in work most of the time, I thank her for supporting me for these years.

To

my family

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Very large-scale integration (VLSI) is the process of fabricating thousands of semiconductor components and interconnections at once by a common set of manufacturing steps. With the rapid advances in VLSI technology, a major problem, one which is growing in importance, is testing. The problems associated with testing of VLSI circuits have been exacerbated by the growing number of circuits placed on an individual chip. With little or no increase in the number of input/output (I/O) pins, more logic must be accessed with almost the same number of I/O pins, making it much more difficult to test a VLSI chip.

As a result of growing circuit complexity, testing is taking an increasingly larger proportion of total product cost. Ironically, the very software design tools that make it possible to put more circuits on a chip at a reduced cost are effectively increasing the cost of circuit testing. The advantages of VLSI are reduced system cost, better performance, and great reliability. These advantages would be lost unless VLSI devices can be tested economically.

1.1 Background

Manufacturing a chip consists of fabrication and testing. Testing is required in order to discover defects in a digital system. Design and test development precede manufacture. Test activities are interwoven with the VLSI design process. Architectural design consists of the partitioning of a VLSI chip into realizable blocks. Either the logic should be synthesized in a testable form or the synthesized logic should be analyzed and improved for testability.

Tests for a VLSI circuit are developed in two phases. In the first phase, known as *design verification*, tests are generated to verify logic correctness and timing behavior of the circuit through simulation. For any reasonably large sequential circuit it would be impossible to enumerate all possible input sequences during testing. As a practical compromise, a subset of inputs, considered to be critical by the designer, is used for verification.

The second phase of test generation consists of generating *manufacturing tests.* Manufacturing tests are used to determine if components and interconnections on the chip are fabricated correctly. These tests thoroughly check every node in the circuit and the effect of every fault is propagated to the circuit outputs. Ideally, manufacturing tests must cover all faults that can possibly occur during fabrication. In this thesis, we concentrate only on the second phase of test generation.

In VLSI circuit design, the testing process is referred to as test generation and fault simulation. The goal of test generation is to obtain test vectors of high quality at an affordable cost. The quality of the test vectors is measured by *fault coverage* which is the fraction of the modeled faults detected by the test vectors. Given a set of faults and a set of test vectors, the goal of fault simulation is to determine which faults are detected by the test vectors. Both test generation and fault simulation rank equally in importance and complement one another. Test vectors capable of distinguishing between good circuits and faulted circuits do not become effective until these vectors are simulated so that their effects can be determined. Conversely, extremely accurate simulation with very precise models, and poor test vectors, will not effectively uncover many defects.

There are various factors that contribute to testing and its cost. Testing cost is determined mainly by the cost of real time test pattern generation and test application. Test pattern generation cost depends on the computer time required to run the test pattern generation program. Test application cost is determined by the cost of equipment plus the testing time required to apply the test. This time may be assumed to be directly proportional to the number of tests. For combinational circuits, a test is a test vector. For sequential circuits, a test is a sequence of test vectors.

A straightforward method for determining the testability of a circuit is to use an Automatic Test Pattern Generation (ATPG) program. It generates test vectors and determine the fault coverage. The running time of the program, the number of test patterns generated, and the fault coverage provide a measure of the testability of the circuit.

1.2 Problems Addressed in this Thesis

This thesis presents a new approach for test generation of sequential circuits. First, cover extraction is performed as a preprocess. A new backward assignment method is presented to extract the ON/OFF sets of the primary outputs and next state lines. Then a novel ATPG system is presented to generate test sequences. By disabling all flip-flops in a sequential circuit, the sequential circuit is transformed into a pseudo-combinational circuit. Then an extended transitive closure algorithm extracts the implication graph and the SAT formula from the model of the circuit incorporating necessary conditions for fault activation and path sensitization. State justification and state differentiation are efficiently performed using the ON/OFF sets of the primary outputs and next state lines. To enhance the efficiency of state differentiation in the existing three-phase ATPG, a novel backward deterministic method for state differentiation is proposed. This method generates a compact testing sequence for a given fault.

1.3 Organization of this Thesis

The thesis is organized as follows. In Chapter 2, the test generation terminologies and the fault models are introduced. The testing problems caused by combinational circuits and sequential circuits are addressed.

The previous work in test generation for combinational circuits and sequential circuits is described in Chapter 3.

In Chapter 4, first, observations that initiated this research work in test generation for sequential circuits are given. Then after the steps of cover extraction, combinational circuit test generation, state justification, and state differentiation are briefly introduced, a novel test generation system for sequential circuits is presented. The algorithms used in these steps are described in Chapters 5 - 7 in detail.

A new backward assignment algorithm for cover extraction is described in detail in Chapter 5. It can efficiently extract the ON/OFF sets of the primary outputs and next state lines.

Chapter 6 describes in detail a transitive closure method for pseudo-combinational circuit test generation. A Boolean difference equation is derived from the circuit model incorporating necessary conditions for fault activation and path sensitization. Efficient transitive closure computations are presented.

In Chapter 7, state justification and state differentiation are described. To enhance the efficiency of state differentiation in the existing ATPG system, a new backward deterministic algorithm for state differentiation is developed.

Experimental results with ISCAS'89 benchmarks are presented in Chapter 8. These results are compared to the existing test generation systems. Chapter 9 concludes this thesis.

CHAPTER 2

FAULT MODELS AND TESTING PROBLEMS

In this chapter, the test generation problems are presented. The test fault models are identified and formulated in Section 2.1. Section 2.2 introduces the test generation terminologies used through this work. In Section 2.3, the problems of test generation for combinational circuits and sequential circuits are presented.

2.1 Faults in VLSI Systems

The testing of a digital logic circuit involves the application of stimuli to the circuit and the evaluation of the response to determine whether the circuit is functionally correct. An important part of testing is the creation of effective stimuli. In, practice, the most commonly occurring faults are modeled. The *fault model* is a computer model of the circuit that has been modified to conform to some premise or conjecture about real physical defects. Then, input stimuli are created which can distinguish between the fault-free and the faulted models. There are a number of advantages of this approach [24]:

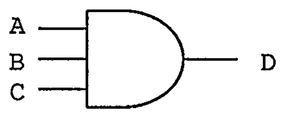
- It is effective to create specific tests for faults most likely to occur.
- The effectiveness of the test set can be measured by determining how many faults can be covered by the set of test vectors.

- Specific defects can be associated with specific test patterns. If a circuit under test responds to a test pattern incorrectly, there is information indicating the faulty component or a set of components.

This method has become a standard approach to developing tests for digital logic failures.

It is desirable to describe faults at various levels of abstraction in VLSI systems. A fault which is described at a very low level, e.g., the transistor level, may accurately describe the physical phenomena causing the fault. One of the difficulties with this level is the tedious task of analyzing each individual component in the circuit. Further complicating the task is the fact that there are several technologies in use and each has its own unique way to perform digital logic operations.

Designers have long used logic symbols to represent their designs. These symbols reduce the complexity of the logic circuit drawings and have the advantage of being technology-independent. Figure 2.1 shows the logic diagram of an AND gate and its truth table.



A	B	C	D
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Figure 2.1. Three input AND gate with its truth table.

With these symbols, the circuits can be logically represented at a higher level, i.e., the gate level. The faults can be described at the gate level and it would be simpler for the purpose of analysis to consider the faults at that level. An important advantage of this representation is the fact that a computer algorithm can be defined upon these logic operators, which are, for most part, independent of the particular technology chosen to implement the circuits.

2.1.1 Fault Models

Fault models are descriptions of the effect of a defect or failure in a circuit. One of the earliest and still widely used fault models at the gate level of abstraction is the *stuck-at* model. In this model, it is assumed that physical defects and faults will result in the lines at the logic gate level of the circuit being permanently stuck at logic value 0 or 1. This model is popular since many defects at the transistor level can be modeled at the gate level.

It is impractical to test for the combinations of all the stuck-at faults in a circuit. This has led to the adoption of the *single-fault* assumption. When attempting to create a test, it is assumed that a single fault exists.

Consider a circuit containing nets which interconnect various components in the circuit. At one time, each net may have only one of the following results:

- Fault-free
- Stuck-at-1, i.e., s-a-1
- Stuck-at-0, i.e., s-a-0

2.1.2 Fault Equivalence and Dominance

In building fault lists, it is often observed that some faults are indistinguishable from others. In Figure 2.1, faults A, B, or C stuck-at 0 would result in the output D being permanently 0 and, therefore, it is impossible to distinguish between an input stuck-at 0 from the output stuck-at 0. These faults are said to be *equivalent*. There is no logic test that can distinguish between them. More precisely, if T_a is the set of tests which detect fault a and T_b is the set of tests which detect fault b, and if $T_a = T_b$, then it is not possible to distinguish a from b.

When we test for inputs, e.g., A, B or C s-a-1, we simultaneously test for the output D s-a-1. A s-a-1 fault on the output, however, prevents one from testing any of the input s-a-1 faults. We say that the output D s-a-1 fault dominates the input s-a-1 fault. In general, fault a *dominates* fault b if T_b is included in T_a. From this definition it follows that if fault a dominates fault b, then any test which detects fault b will detect fault a.

Since computer time for circuit testing is affected by the size of the fault list, the reduction of the fault list, a process called *fault collapsing*, can reduce test generation and simulation time. Therefore, fault equivalence and dominance relations are used to reduce the size of fault lists.

2.2 Testing Terminologies and Definitions

A sequential circuit is shown in Figure 2.2. The circuit consists of a combinational logic block and some feedback flip-flops. The inputs and outputs of flip-flops are the next state and present state lines, respectively. There are p primary inputs, n present state lines, n next state lines, and q primary outputs. Here it is assumed that the present state and next state lines are neither controllable nor observable. The task of test generation for sequential circuits is to find primary input sequences which can propagate the faults in the sequential circuit to the primary outputs.

Figure 2.3 shows a conventional *iterative array model* [5] used in the test genera-

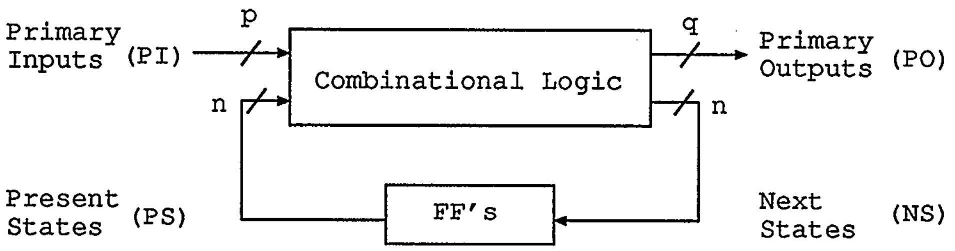


Figure 2.2. A sequential circuit.

tion of sequential circuits. Assume there is a fault, F, in the combinational logic block of the sequential circuit shown in Figure 2.2, we duplicate the block in terms of each clock cycle, i.e., time-frame. The iterative array in Figure 2.3 is logically equivalent to the sequential circuit shown in Figure 2.2. If an input sequence $PI^1, PI^2, \cdots, PI^k$ is applied to the sequential circuit in initial state PS^1, i.e., a reset state, it generates an output sequence $PO^1, PO^2, \cdots, PO^k$ and the next state sequence $NS^1, NS^2, \cdots, NS^k$ ($PS^{i+1} = NS^i$, $1 \le i < k$).

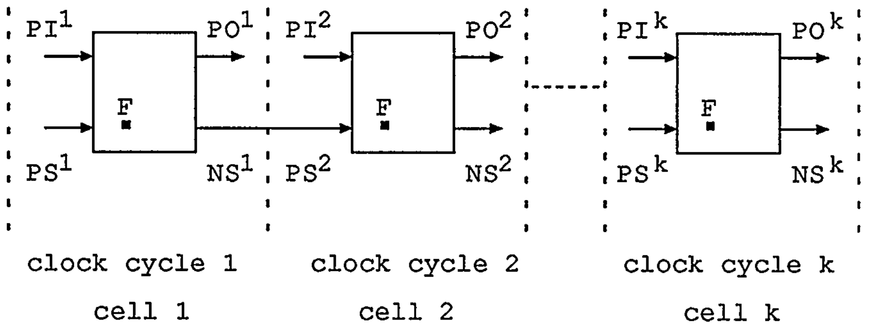


Figure 2.3. An equivalent pseudo-combinational iterative array to the sequential circuit in Fig. 2.1.

Definition 2.2.1 Beginning with the present state in clock cycle 1, PS^1, we set the reset state values and wish to produce an input sequence, $PI^1, PI^2, \cdots, PI^k$, which, when applied to clock cycles $1, 2, \cdots, k$, propagates the effect of the fault F to the primary outputs, PO^k, during the kth clock cycle. This input sequence is called a *test sequence* for the fault.

Unlike combinational circuits, which only needs an input test vector to test a fault, a sequential circuit may require a test sequence of up to 2^{n+1} input test vectors, where n is the number of memory elements (flip-flops) in the sequential circuits [5].

In sequential circuit testing, a *state* is a bit vector. Its length is equal to the number of memory elements in the sequential circuit. In general, a state is a *cube*, i.e., the values at the different bit positions may be 0, 1 or X (don't care). A *minterm* state is a state with only 0's or 1's as bit values. A cube state is a group of minterm states. A *universal cube* is a cube with all X entries.

Definition 2.2.2 State S_1 *implicates* state S_2, if and only if, every state contained in S_1 is also contained in S_2. That is, state S_2 *covers* state S_1.

For example, state (0, 1, 0) is a minterm state, and state (0, 1, X) is a cube state. There are two minterm states (0, 1, 0) and (0, 1, 1) in the state (0, 1, X), so state (0, 1, 0) implicates state (0, 1, X). (X, X, X) is a universal cube.

The sequential circuits discussed here are assumed to have a *reset state.* All test sequences are applied to the sequential circuit with the reset state as the starting state. Some faults in the circuits may be *redundant*, i.e., their existence does not change the behavior of the circuit. There are two kinds of redundant faults, combinational redundant and sequential redundant.

Definition 2.2.3 A *combinational redundant* fault cannot be propagated to the primary outputs or the next state lines, beginning from any state, with any input vector.

Definition 2.2.4 A *sequential redundant* fault cannot be excited or whose effect cannot be propagated to the primary outputs using any sequence of input vectors starting from the reset state.

Definition 2.2.5 An *excitation vector* for a fault is an assignment that propagates the fault to either the primary outputs or the next state lines. This assignment consists of two parts, the primary input and the present state. The present state of an excitation vector is called an *excitation state.* The primary input of an excitation vector is an *excitation input.*

Definition 2.2.6 The process of finding an input sequence which takes a circuit from the reset state into the excitation state is called *state justification.* The corresponding input sequence is a *justification sequence.*

There are two kinds of state justification, *forward* state justification and *backward* state justification. In the forward state justification, the search is done from the reset state to the excitation state; and vice versa for the backward state justification. If the excitation vector propagates the fault to the next state lines, *state differentiation* is required.

Definition 2.2.7 *State differentiation* is the process of propagating the effect of the fault on the next state lines to the primary outputs. A *differentiation sequence* for a pair of states, true state S^T and faulty state S^F, which are different in at least one bit, is an input sequence such that, if the circuit is initially in S^T, the last vector in the sequence produces a different logic value in at least one primary output than if the circuit were initially in S^F.

In circuit testing, the complete test sequence is obtained by combining the justification sequence, the excitation vector, and the differentiation sequence.

When all flip-flops in a sequential circuit are disabled, the sequential circuit becomes a *pseudo-combinational* circuit as shown in Figure 2.4. The primary inputs and present state lines are considered as the inputs of the pseudo-combinational circuit. The primary outputs and next state lines are the outputs of the pseudo-combinational circuit.

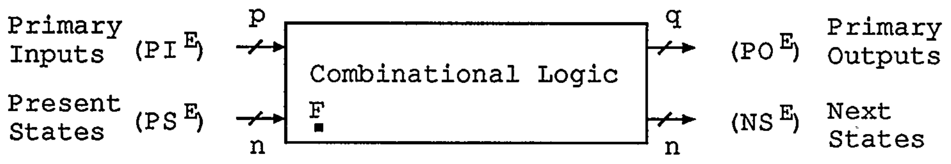


Figure 2.4. A general pseudo-combinational circuit.

Definition 2.2.8 The *output cone* of a primary output is a portion of circuit which includes the primary output and its subtree from the primary output to the primary inputs and present state lines. Any fault site in this subtree is a *node* in the output cone. The *next state cone* of a next state line is a portion of circuit which includes the next state line and its subtree from the next state line to the primary inputs and present state lines.

Consider the sequential circuit, shown in Figure 2.2. Assuming a fault site in the circuit is a node in the output cones of r primary outputs, then we refer to the output cones of these r primary outputs as the *primary output fault region* for the fault under test. Similarly, if a fault site in the logic circuit is a node in the next state cones of s next state lines, the next state cones of these s next state lines compose the *next state fault region* for the fault under test.

To illustrate the idea of a circuit cone, we use a simple sequential circuit s27 from the ISCAS'89 benchmarks. The circuit is shown in Figure 2.5. There is only one primary output G17, and its output cone is shown in Figure 2.6. There are three next state lines G10, G11, and G13. The next state cone of G11 is shown in Figure 2.7. Considering a fault on G15, as G15 is a node in the output cone of the primary output G17, its primary output fault region is the output cone of G17 shown in Figure 2.6. Though G15 is a node in the next state cones of the next state lines G11 and G10, if the fault is to propagate to G10, it must propagate to G11 first. One need only consider the next state line G11. The next state fault region for the fault at node G15 is the next state cone of G11, as shown in Figure 2.7.

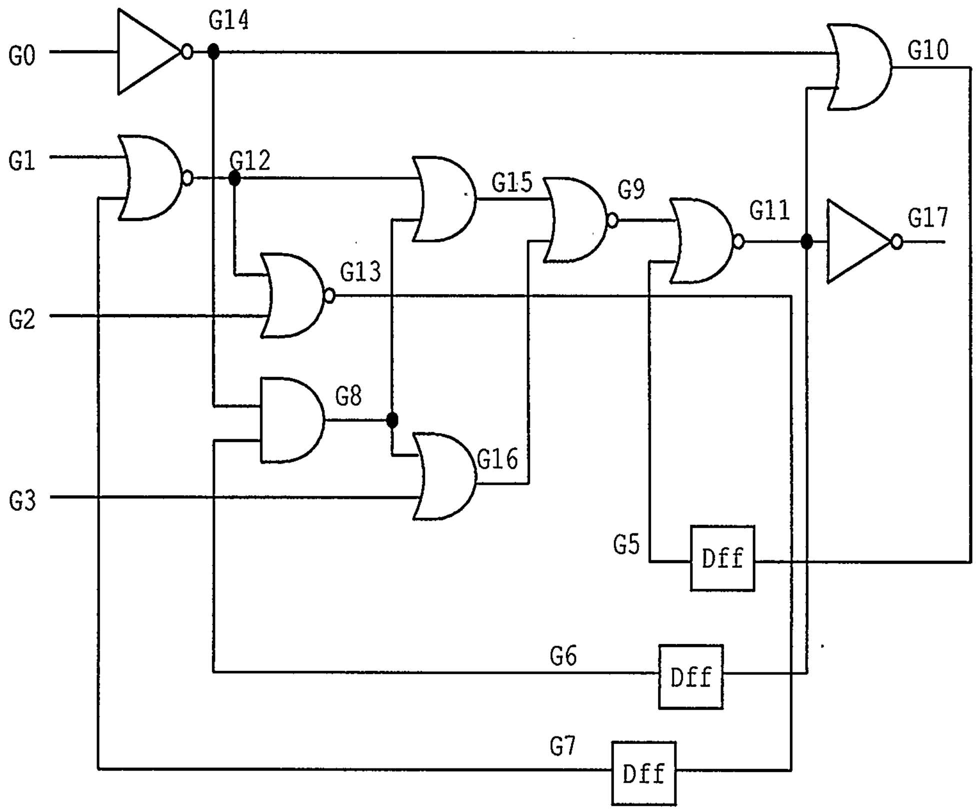


Figure 2.5. Example circuit s27 from ISCAS'89 benchmarks.

Definition 2.2.9 The *ON set* of an output is the complete set of the input values which produce the output logic value 1. The *OFF set* is the complete set of the input values such that the corresponding output is at logic value 0.

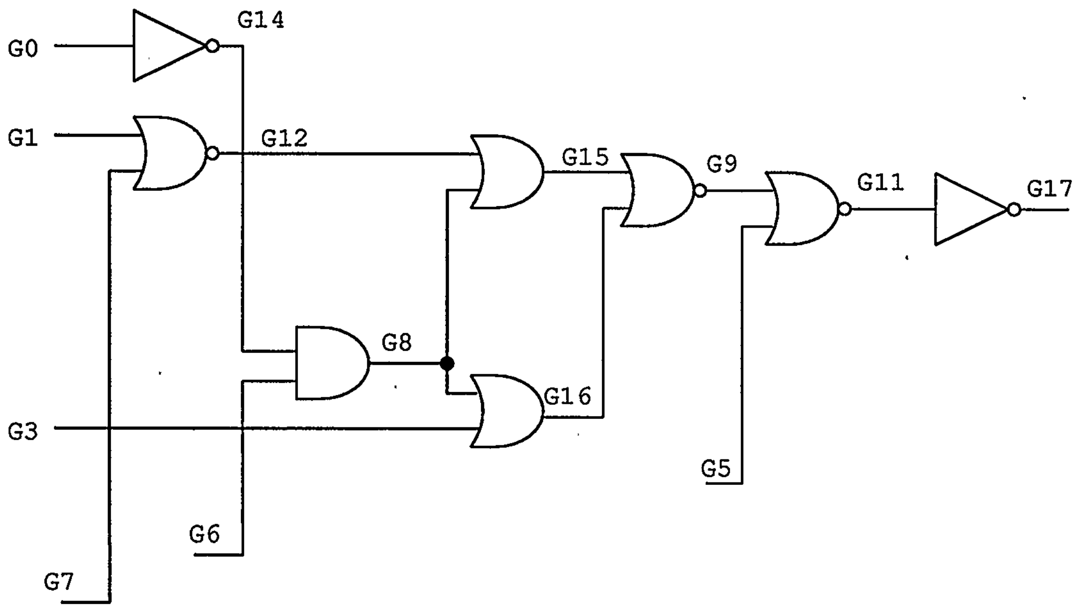


Figure 2.6. The output cone of G17 in circuit s27.

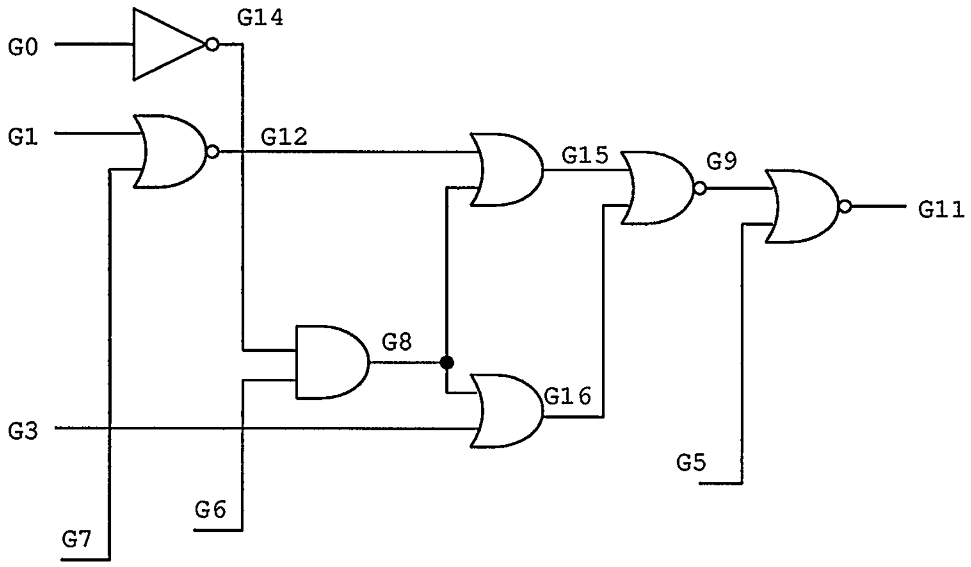


Figure 2.7. The next state cone of G11 in circuit s27.

The process of extracting the ON/OFF sets of the primary outputs and next state lines is called *cover extraction.*

Definition 2.2.10 If a sequential circuit can reach a state F_{out} during the next clock cycle from a state F_{in}, state F_{in} is said to be a *fan-in state* of state F_{out}, and state F_{out} is a *fanout state* of state F_{in}.

All fan-in states of a state can be easily obtained by cube intersection on the ON and OFF sets of the next state lines.

Definition 2.2.11 The *intersection* of two cubes c and d, denoted $c \cap d$, is the set of states that belong to both c and d.

$$c \cap d = \begin{cases} \phi, \text{ if there exists one k}, c_k \cap d_k = \phi, \text{ otherwise} \\ \{(c_1 \cap d_1)(c_2 \cap d_2) \cdots (c_n \cap d_n)\} \end{cases}$$

The intersection of the three value tuple is defined in Table 2.1, where ϕ is the empty set.

Table 2.1. Cube intersection operation.

			d_i	
	$\cap$	0	1	X
	0	0	ϕ	0
c_i	1	ϕ	1	1
	X	0	1	X

Definition 2.2.12 The *sharp product* of two cubes, i.e., $c\#d$, is the set of states that belong to c but not to d.

$$c\#d = \begin{cases} c, \text{ if there exists one k, } c_k\#d_k = \phi; \\ \phi, \text{ if } c_k\#d_k = \varepsilon, \text{ for all k; else} \\ \cup_k\{c_1c_2...c_{k-1}\alpha_kc_{k+1}...c_n\} \end{cases}$$

where $c_k\#d_k = \alpha_k \in 0, 1,\ k = 1, 2, ..., n$.

The sharp product of the three value tuple is obtained in Table 2.2, where ε means implication.

Table 2.2. Sharp product operation.

		d_i		
	#	0	1	X
	0	ε	ϕ	ε
c_i	1	ϕ	ε	ε
	X	1	0	ε

Definition 2.2.13 A *graph* $G = (V, E)$ consists of a finite, nonempty set of *vertices* *V* and a set of *edges E*. If the edges are ordered pairs (v, w) of vertices, then the graph is said to be *directed*; v is called the *tail* and w the *head* of the edge (v, w).

Definition 2.2.14 A *path* is a sequence of edges of the form (v_1, v_2), (v_2, v_3), ..., (v_{n-1}, v_n). We say that the path is *from* v_1 *to* v_n and is of *length n-1.* A *cycle* is a simple path of length at least 1 which begins and ends at the same vertex.

If a graph contains a cycle, it is cyclic; otherwise it is acyclic. A *Directed Acyclic Graph* (DAG) can be used to describe a circuit.

Definition 2.2.15 The *transitive closure* of G is defined as a graph $G*$ which has the same vertex set as G, but has an edge from v to w if and only if there is a path from v to w in G.

Definition 2.2.16 The edges V can be partitioned into equivalence classes V_i, $1 \leq i \leq r$, such that vertices v and w are equivalent if and only if there is a path from v to w and a path from w to v. The graphs $G_i = (V_i, E_i)$ are called the *strongly connected components* of G.

The goal of the satisfiability (SAT) problem [9] is to determine whether there exists an assignment of truth values to a set of variables $(x_1, x_2, ..., x_m)$ that makes the following Boolean formula satisfiable:

$$c_1 \cdot c_2 \cdot ... \cdot c_n \tag{2.1}$$

where $\cdot$ is a logic *and* connector and $c_1, c_2, ..., c_n$ are n distinct clauses. Each clause consists of only literals combined by just logic *or* $(+)$ connector (a literal is a variable or a single negation of a variable).

2.3 Test Generation and Its Problems

With the progress of the VLSI technology, the problem of fault detection for logic circuits is becoming increasingly difficult.

2.3.1 NP-Completeness of Combinational Test Generation

A significant theoretical study by Ibarra and Sahni [17] shows that test generation for combinational circuits belongs to the class of NP-complete problems. This strongly suggests that no test generation algorithm with a polynomial time complexity is likely to exist. The problem of combinational circuit test generation can be viewed as a finite space search problem [12]. For a combinational circuit with m primary inputs, there exists 2^m combinations of input assignments.

In practice, test generation algorithms for combinational circuits appear to be able to achieve lower *average* time growth by using heuristic search techniques. Up to now, some well-known test generation algorithms for combinational circuits, such as D-algorithm [26], PODEM [12], FAN [10], NEMESIS [19], and TRAN [7], have been developed. Some of them perform well for certain circuit structures.

2.3.2 Test Problems in Sequential Circuits

Test generation for sequential circuits has long been recognized as a difficult task [5, 23]. It remains to be a challenge in spite of a history of attempts dating back to the late 1960s. One new factor which complicates the task of creating tests for sequential circuits is the presence of memory elements.

For combinational circuits, it is possible, but not necessarily reasonable, to create a complete test for logic faults by applying all possible binary combinations to the inputs. This is not true for sequential circuits with memory elements. Not only may they requires more than 2^m tests, they are also sensitive to the *order* in which stimuli are applied. It has been shown [5] that a fault in a general synchronous sequential circuit may require a test sequence of up to 2^{n+1} input test vectors, where n is the number of memory elements in the sequential circuits. This shows that the search space for sequential circuit test generation is very large.

2.4 Summary

The cost of manufacturing a VLSI chip is very much affected by the testing cost for the chip. The large number and complex nature of potential physical failures suggests that a practical approach to testing should avoid working directly with the physical failure. One approach for solving the problem is to describe the effects of physical failures at some higher levels of abstraction. The stuck-at fault model at the gate level is the most popular model describing present VLSI testing methodology.

It has been recognized that test generation for sequential circuits is an extremely difficult problem. Different approaches have been used to tackle the test generation problem for sequential circuits, either by randomly generating test sequences or by using other deterministic test generation methods. It is a challenging topic to develop an efficient test generation system for VLSI sequential circuits.

CHAPTER 3

PREVIOUS WORK

In this chapter, the previous work in test generation for combinational circuits and sequential circuits is presented.

3.1 Test Generation for Combinational Circuits

Existing ATPG systems for combinational circuits fall into two classes: structural methods, such as PODEM [12], and algebraic methods. Structural search methods use a data structure representing the circuit to be tested. To generate a test pattern, they assign values that cause a discrepancy at the faulted line (*fault site*) and then search for consistent values for all circuit lines such that the discrepancy is visible at a circuit output.

Among structural search methods, the D-algorithm, developed by Roth [26], is probably the most known test generation algorithm. This algorithm adopts a five-valued 0, 1, $X, D, \overline{D}$ calculus to be able to carry out the sensitization and the line justification procedures in a very formal manner. The faulty line is assigned a D, or $\overline{D}$ depending on the fault on the line. The calculus and the circuit structure information are used to determine values on the other lines so that the D or $\overline{D}$ can be sensitized to the primary outputs. A line justification step is then carried out to justify the values assigned in the preceding step. Both the sensitization and the line justification steps may have to be carried out many times before a test vectors is obtained.

A class of circuits for which the D-algorithm performs particularly poorly are those containing exclusive-or trees. The degradation in performance arises due to excessive amount of backtracking. This observation motivated Goel [12] to devise a new test generation algorithm called *path oriented decision making* (PODEM). He used a branch and bound technique. The algorithm starts by assigning a value of 0 or 1 to a selected primary input (PI) line, and then determines its implication on the propagation of D or $\overline{D}$ to a primary output. If no inconsistency is found, it again somehow selects another PI line and, assigns a 0 or 1 to it, and then repeats the process, which is referred to as *branching*. If an inconsistency is determined in the branching, the branching stops and bounding starts. The PI line which was most recently assigned a binary value is assigned the complimentary value, and branching starts again. The complete process stops when either a test vector is found or when the fault is determined to be undetectable. PODEM implementations are known to run an order of magnitude faster than the D-algorithm on most circuits.

Fujiwara and Shimono [10] described techniques to further accelerate a path-sensitization algorithm like PODEM. Their algorithm, called FAN, does extensive analysis of the circuit connectivity in a preprocessing step to minimize backtracking. FAN has employed a better heuristic in the bounding-and-branching steps to speedup the test generation process.

In these structural methods, backtracking, which is a branch procedure terminated by a bound step, is the most computationally expensive step in the process of searching for a test vector. The branching step goes as deep in the binary search tree as possible, while the bound step backs up in the binary search tree to the most recent node with an unused alternative assignment.

Instead of performing a search on a data structure representing a circuit, algebraic methods produce an equation describing all possible tests for a particular fault and then simplify the resulting equation. A typical algebraic method is the Boolean difference method, proposed by Sellers et al. [27]. Once the Boolean difference formula for the testing problem is obtained, it is simplified by using the basic laws of Boolean algebra or using identities specific to the Boolean difference. The tedious nature of the algebraic manipulations involved in solving formulae using the Boolean difference led to its disfavor as a practical tool for test pattern generation [24].

Recently, Larrabee [19] proposed a Boolean satisfiability (SAT) method for generating test vectors for single stuck-at faults in combinational circuits. This new method generates test vectors in two steps. First, it constructs a formula expressing the Boolean difference between the unfault and faulted circuits. Second, instead of performing symbol manipulation, it applies a SAT algorithm to satisfy the formula. This new method has, in practice, produced excellent results for the problem of combinational circuit test generation.

Later, Chakradhar, Agrawal, and Rothweiler [7] developed a transitive closure algorithm for combinational circuit test generation. A test is obtained by determining signal values that satisfy a Boolean difference equation derived from the model of the circuit incorporating necessary conditions for fault activation and path sensitization. The method is a sequence of two main steps that are repeatedly executed: transitive closure computation and decision-making. The transitive closure contains global pairwise (or binary) logical relationships among all signals. Higher-order signal relationships are represented as additional ternary relations. A key feature of the algorithm is that signal dependencies derived from the transitive closure are used to reduce ternary relations to binary relations that in turn dynamically update the transitive closure. The signals are either determined from the transitive closure or are enumerated until the Boolean equation is satisfied. The transitive closure algorithm has produced excellent results on popular test pattern generation benchmarks.

3.2 Test Generation for Sequential Circuits

The Earlier algorithms represented sequential circuits as iterative combinational circuits. Some test generation algorithms for combinational circuits were extended to test sequential circuits [18, 25]. An algorithm that implements this method has been programmed into a commercial package called LASAR [29]. Several approaches [22, 28] based on the extensions of the classical D-algorithm were presented to solve the problem of test generation for sequential circuits. Shteingart et al. [28] gave an efficient technique for modeling sequential components. Although some progress was made in these attempts, an effective solution for circuits with more than a few hundred gates and large sequential depths was not available at that time.

Due to the relative ineffectiveness of these ATPG systems, many large digital systems are being designed in compliance with design-for-testability rules which attempt to reduce the complexity of the test problem. The object of design-for-test is to provide guidelines which insure the creation of testable designs. A popular approach is to make the memory elements controllable and observable, i.e., a scan design [1]. The flip-flops and/or latches are designed to be able to operate in either parallel load or serial shift mode. In the normal mode of operation, flip-flops and latches are configured for parallel load. For testing purposes the flip-flops are switched to a serial shift mode. In serial mode, any needed test values can be loaded by serially clocking in the desired values. In similar fashion, any values present in the flip-flops can be observed by clocking out their contents while in the serial shift mode. Scan design approaches have been successfully used to reduce the complexity of the problem of sequential circuit test generation by transforming the problem into that of combinational circuit test generation. However, in some cases, the cost in terms of area and/or performance and/or extra numbers of I/O pins is unaffordable.

Recently, considerable progress has been made in test generation for sequential circuits. A heuristic, simulation-based test generation algorithm was presented by Agrawal et al. [2]. Ma, Devadas, Newton, and Sangiovanni-Vincentelli [21] described a PODEM-based deterministic approach to sequential circuit test generation, called STALLION. It first extracts a partial state transition graph (STG) of a sequential circuit. The construction of the partial STG is based on an efficient state-enumeration algorithm that aims at finding paths from the reset state to different valid states (states reachable from the reset state) in the STG. Then test sequences for line stuck-at faults can be generated using the two-phase ATPG system: fault excitation and propagation, and state justification.

Later, a new system, STEED, was proposed by Ghosh, Devadas, and Newton [11] to improve STALLION. STEED decomposes the problem of sequential circuit test generation into three subproblems, i.e., excitation vector generation, state justification, and state differentiation. Given a fault under test, it first generates a combinational excitation vector that propagates the effect of the fault to the primary outputs or the next state lines. Combinational circuit test generation is based on a PODEM-based algorithm. A justification step is then performed, which involves finding a justification sequence for the excitation state. This step is carried out using a sequence of cube intersections on the complete or partial ON/OFF-sets of the next state lines. Thus a justification sequence is found. If the effect of the fault has been propagated to the next state lines alone, the true-faulty state pair is produced by the excitation vector. A differentiation sequence for this true-faulty state pair is obtained using another sequence of cube intersections, this time using the ON/OFF-sets of the primary outputs. It is shown that this three-phase ATPG for sequential circuits is an efficient method. STEED significantly improved STALLION in terms of computing time for the same fault coverage.

Cho, Hachtel, and Somenzi [8] have recently given an efficient algorithm, VERITAS, for sequential circuit test generation. VERITAS is based on implicit state enumeration and a three-phase ATPG. The approach identifies sequential redundancies through reachability analysis of sequential circuits. It constructs the product machine of two sequential circuits to be compared. The reachability analysis is performed by traversing the finite state machine to find any difference in I/O behavior. When an output difference is detected, the information obtained by reachability analysis is used to generate a test sequence. As the product machine traversal (PMT) is quite resource-demanding, a three-phase ATPG system is used first to deal with most of the faults. PMT is used only for the faults for which the three-phase ATPG fails to generate test sequences. VERITAS further improved STEED in terms of running time, test vector length, and fault coverage. It is difficult, however, for VERITAS to handle large size sequential circuits.

These approaches are capable of generating tests for sequential circuits with 1000-3000 gates. Due to the difficulty of test generation for sequential circuits, significant improvements are needed for the testing of larger sequential circuits.

3.3 Summary

Up to now, some well-known test generation algorithms for combinational circuits have been developed and perform well for certain circuit structures. Existing ATPG systems for combinational circuits fall into two classes: structural and algebraic methods. Both Boolean satisfiability and transitive closure methods have produced excellent results on popular test pattern generation benchmarks.

For sequential circuit test generation, some progress has been made in the past several years. The three-phase ATPG system is shown to be an efficient method. Due to the difficulty of test generation for sequential circuits, significant improvements are needed for very large scale sequential circuits.

CHAPTER 4

A NOVEL TEST GENERATION SYSTEM FOR SEQUENTIAL CIRCUITS

In this chapter, we present an efficient test generation algorithm for sequential circuits. A transitive closure algorithm has been developed for combinational circuit test generation. We extend the transitive closure algorithm to test generation of sequential circuits. To make the previous three-phase ATPG system more efficient, a new backward deterministic method for state differentiation is developed. This algorithm offers significant efficiency improvements for test generation of large sequential circuits.

At first, observations that initiated this research work in sequential circuit test generation are given. Then after briefly introducing the steps of cover extraction, pseudo-combinational circuit test generation, state justification, state differentiation, fault simulation, and determination of redundant faults, a novel test generation system for sequential circuits is presented. The algorithms used in cover extraction, pseudo-combinational circuit test generation, state justification, and state differentiation will be described in Chapters 5 - 7 in detail.

4.1 Observations

Up to now, the most popular ATPG systems for sequential circuits use the three-phase ATPG method: excitation vector generation, state justification, and state differentiation. The first phase, in the most cases, uses a PODEM-based combinational

ATPG, such as STALLION [21] or STEED [11]. This phase usually takes a large fraction of the total test generation time. As indicated in Section 3.1, Boolean satisfiability (SAT) method [19] and transitive closure method [7] have been developed to perform test generation for combinational circuits. Both SAT approach and transitive closure algorithm have obtained superior results over the PODEM-based algorithms. In our approach, we use the transitive closure algorithm to perform the first phase, i.e., excitation vector generation, in sequential circuit test generation.

The second and third phases are state justification and state differentiation. They usually take a small fraction of the total test generation time. State differentiation in the existing three-phase ATPG systems lacks efficiency in dealing with the unspecified inputs in excitation vector and justification sequence. So STEED has to apply all possible assignments to the unspecified inputs before it concludes that a test for the fault under consideration does not exist. There exists 2^n possible minterm states for n unspecified inputs. Considering that each possible minterm state may need to perform state justification, the real search space is much larger than 2^n.

In this thesis, we propose a new backward deterministic method for state differentiation. In our approach, cubes, rather than minterm states, are used to represent states. Instead of using minterm state differentiation, our method searchs backward to specify the cubes into real excitation states. This has considerably reduced the running time for sequential circuit test generation.

4.2 A Novel Test Generation System for Sequential Circuits

The system starts by extracting the ON/OFF sets of the primary outputs and next state lines. A new backward assignment method is proposed to perform cover extraction. We employ the three-phase ATPG approach to generate test sequences.

The transitive closure algorithm is extended to test generation for sequential circuits. The problem of test generation for sequential circuits is decomposed into three subproblems:

- pseudo-combinational circuit test generation: All flip-flops in the sequential circuit are disabled, and sequential circuit test generation becomes combinational circuit test generation. The transitive closure algorithm is used to find the excitation vector for the pseudo-combinational logic circuit.
- state justification: An input sequence is found to take a circuit from the reset state into the excitation state.
- state differentiation: An input sequence is found to propagate the effect of the fault on the next state lines to the primary outputs.

In the following discussion, we will describe briefly cover extraction, transitive closure based pseudo-combinational circuit test generation, state justification, state differentiation, fault simulation, and determination of redundant faults. Then the new test generation system for sequential circuits is presented.

4.2.1 Cover Extraction

The objective of cover extraction is to extract the ON/OFF sets of the primary outputs and next state lines. At first, all flip-flops in a sequential circuit are disabled. The sequential circuit becomes a pseudo-combinational circuit. The inputs of all flip-flops (next state lines) and the primary outputs are considered as outputs of the combinational circuit. The outputs of all flip-flops (present state lines) and the primary inputs are considered as inputs of the combinational circuit.

For each output of the pseudo-combinational circuit, the ON/OFF sets are extracted by assigning the corresponding output line to logic value 1 or 0 and using a new and efficient backward assignment method to implicitly enumerate the input combinations that can set the output line to 1 or 0. A similar backward assignment method has been successfully used in test generation for combinational circuits [30].

At first, the combinational circuit is represented as separate output cone for each output. For each circuit cone, we assign the output of the cone to 1 or 0. Then we propagate the assignment backward to the inputs of the cone. Finally, the combination of the assignments at the inputs of the cone is the ON or OFF set of the output.

Due to the connectivity of the logic circuit, some nodes in the circuit may be assigned more than once. Therefore with the increase of the circuit's depth, the number of assignments for each node per level may increase dramatically. So the CPU time for generating the ON/OFF sets may grow dramatically. A simple method is to set a limit for the maximum number of the assignments at each node [30]. Limiting the maximum number of assignments per node can dramatically decrease the extraction time. But after setting the limit, the ON/OFF sets obtained may be incomplete.

We use a different and efficient method. After the number of assignments reaches the limit, we use logic minimization to compress the assignments. The method is based on the fact that each ON/OFF set usually requires less than a few hundred vectors for most of ISCAS'89 benchmark sequential circuits after logic minimization. This method assures that we obtain complete ON/OFF sets of the primary outputs and next state lines. Also it makes the storage of ON/OFF sets memory efficient. For some large circuits, it might not be possible to generate the complete cover. This method would extract as many vectors in the cover as possible. The ON/OFF sets are saved in bit vectors, which are similar to those used in ESPRESSO [4].

Three methods can be used to extract the ON/OFF sets.

1) The output of the circuit is set to 1 or 0, and the backward assignment method is used to generate the ON or OFF set separately.

2) Because the ON set and OFF set for a same output are complementary, which means that the union of the ON set and OFF set for the same output should correspond to the universal cube, when we generate the ON set, the OFF set can be easily obtained by disjointing the ON set from the corresponding universal cube.

3) The output of the circuit can be set to the logic value D or $\overline{D}$. The backward assignment method is used to generate the D set. When D is equal to 1, the ON set is obtained. When D is 0, the OFF set is obtained. A part of backward assignment rules of value D can be found in [30].

We use the first method to generate the ON/OFF sets of the primary outputs and next state lines. The backward assignment method will be discussed in Chapter 5 in detail.

4.2.2 Pseudo-Combinational Circuit Test Generation

Our current method considers one fault at a time. Given a fault for which that a test sequence is to be generated, the first step in test generation for sequential circuits is to generate a combinational test vector in the pseudo-combinational circuit for the fault. Figure 2.4 shows a pseudo-combinational circuit obtained from a general sequential circuit by disabling all flip-flops. The goal of test generation for a pseudo-combinational circuit is to find an excitation vector (PI^E, PS^E) which excites the fault to PO^E or NS^E.

Our test generation algorithm for combinational circuits, which will be described in Chapter 6 in detail, is based on a transitive closure method [7]. At first, the algorithm tries to propagate the effect of the fault to the primary outputs. If failed, the algorithm tries to propagate the effect of the fault to the next state lines. When the fault is combinational redundant, the effect of the fault cannot propagate to either the primary outputs or the next state lines.

To make state justification easier, the excitation vector is generated with as many don't care entries as possible – some lines may be left unknown. If the excitation state can not be justified, a new excitation vector should be generated. The new vector should be disjointed from all the previous states. This assures that all new generated excitation states are not used previously.

We notice that each fault may only be a node in circuit cones of some primary outputs and/or next state lines. To generate excitation vector efficiently, we only need to consider the related part of the circuit with the fault. When the algorithm tries to propagate the effect of the fault to the primary outputs, we search the circuit forward from the fault site to the primary outputs and find all related primary outputs. The output cones of these primary outputs compose the primary output fault region for the fault. Similarly, when the algorithm tries to propagate the effect of the fault to the next state lines, we should search for an excitation vector in the next state fault region for the fault. Because the fault region is smaller than the original circuit, the search effort and time are decreased.

4.2.3 State Justification

Once a combinational excitation vector is found for a fault in the pseudo-combinational circuit with as many don't care entries as possible, state justification is used to justify if the excitation state is reachable from the reset state. Usually the excitation state is a cube. If the reset state implicates the excitation state, the fault can be excited from the reset state. If not, the excitation state should be justified by using state justification.

The iterative array model in Figure 4.1 is used to illustrate state justification. The excitation input PI^E and excitation state PS^E excite the effect of a fault under test to PO^E or NS^E. As the sequential circuit discussed here is assumed to have a reset state, all valid states begin from this reset state. The goal of state justification is to find an input sequence $PI^{J1}, PI^{J2}, \cdots, PI^{Jk}$ which places the sequential circuit into the excitation state PS^E from the reset state. If PS^{J1} is the reset state, the justification sequence $PI^{J1}, PI^{J2}, \cdots, PI^{Jk}$ is found. The set of states traversed during state justification, $PS^{J1}, PS^{J2}, \cdots, PS^{Jk}$, constitute the justification path.

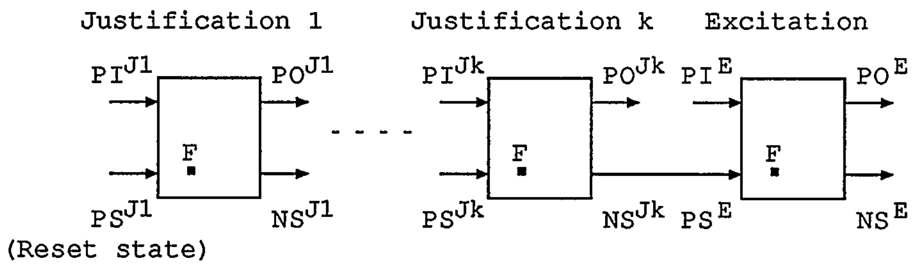


Figure 4.1. General iterative array model for state justification.

State justification can also be illustrated by the state transition graph (STG) shown in Figure 4.2. PS^E is the excitation state, and we need to find a justification path from the reset state to the state PS^E.

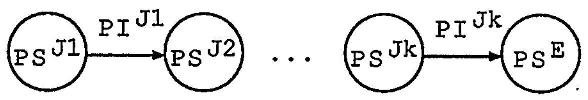


Figure 4.2. State transition graph for state justification.

There are two methods to perform state justification, forward and backward, depending on whether the search is conducted from the reset state to the excitation state or vice verse. Here we use backward state justification. All fan-in states of the excitation state are obtained by performing cube intersection on the corresponding ON/OFF sets of the next state lines. If the reset state implicates the fan-in states, a single vector justification sequence is found. Otherwise, the process is repeated for the fan-in states being currently justified to try to find multi-vector justification sequence.

It is noted that all fan-in states obtained in state justification are cubes. Because a cube state is a group of minterm states, using cubes is helpful to find a shorter justification sequence. Thus the justification time is reduced and the quality of the test pattern generator increases. Once the justification sequence is found, fault simulation is used to check if the excitation state is justified. The algorithm of state justification is given in Section 7.1.

4.2.4 State Differentiation

If the combinational excitation vector propagates the fault to the primary outputs, and the excitation state is justified, a test sequence for the fault is successfully generated. However, if the combinational excitation vector propagates the fault to the next state lines, state differentiation is required to continually propagate the effect of

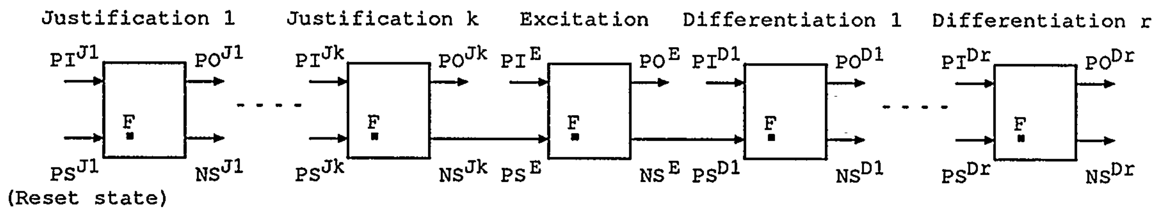


Figure 4.3. Iterative array model for state differentiation.

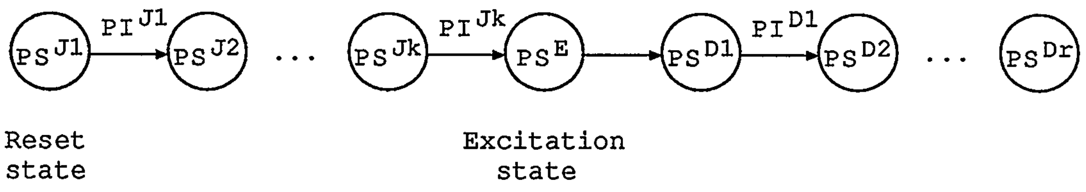


Figure 4.4. State transition graph for state justification and state differentiation.

the fault to the primary outputs.

The iterative array model in Figure 4.3 is used to illustrate state differentiation. The excitation input PI^E and excitation state PS^E excite the effect of a fault under test to the next state lines NS^E, and a justification path from PS^{J1} (reset state) to PS^{Jk} is found. The goal of state differentiation is to find an input sequence $PI^{D1}, PI^{D2}, \cdots, PI^{Dr}$ which propagates the effect of the fault on the next state lines of the excitation state clock cycle to the primary outputs of the rth differentiation clock cycle. The primary input sequence $PI^{D1}, PI^{D2}, \cdots, PI^{Dr}$ is the differentiation sequence. The set of states traversed during state differentiation, $PS^{D1}, PS^{D2}, \cdots, PS^{Dr}$, constitutes the differentiation path. The test sequence is obtained by concatenating the justification sequence, the excitation vector, and the differentiation sequence. The state transition graph shown in Figure 4.4 is used to illustrate the test sequence.

To reduce the time of state differentiation, random state differentiation is performed first. If random state differentiation fails, a deterministic state differentiation algorithm is used.

At first, fault simulation is used to create the true and faulty states (S_1^T, S_1^F) with the combinational excitation vector. By employing cube intersection on the ON and OFF sets of each primary output, we try to find an input vector which produces a different output on the corresponding primary output, beginning from the true and faulty states separately. Such an input vector constitutes a single-vector differentiation sequence. If a single-vector differentiation sequence cannot be found, all the fan-out states of the true and faulty states are found via repeated cube intersection. This is performed by finding an input vector that produces a different output on at least one next state line for the true and faulty states with the ON/OFF sets of each next state line. If the input vector is found, a new pair of true and faulty states (S_2^T, S_2^F) are obtained. For the new true and faulty states, a single-vector differentiation sequence is sought again. If found, a two-vector differentiation sequence is constructed. Otherwise, a pair of states fanning out from some fan-out state pair is picked and differentiation between this pair is attempted. The process continues until a differentiation sequence is found or there does not exist any differentiation sequence for S_1^T and S_1^F. Once the differentiation sequence is found, the entire test sequence is fault simulated to check if the fault can be detected.

As with state justification, in the general case, state differentiation is attempted between disjoint groups of states (cube states) rather than a minterm state pair. This means that some bits in the true and faulty states are unknown. The existence of a differentiation sequence between two groups of states means that if *any* state A from the true group is chosen, along with a *corresponding* state A' from the faulty group, then the differentiation sequence will be able to differentiate between the state A and A'. Since this is a strong requirement, it is often impossible to find a differentiation sequence between the state groups [11]. This does not means that a test for the fault does not exist. In order to find a test, usually it is necessary to set some unspecified bits in the primary inputs or the present states of the justification sequence and excitation vector to either 0 or 1. A simple method can be applied where the excitation state is separated into a group of minterm states, and state justification and differentiation are performed on the minterm states. The disadvantage of this method is the long running time.

A novel and efficient backward deterministic method is developed in this thesis to solve this problem. After the combinational excitation state is found to propagate the fault to the next state lines with as many don't care entries as possible and is justified successfully, the backward deterministic method for state differentiation is used. When we search forward to perform state differentiation, if some unspecified bits in the present states and the primary inputs of the whole sequence are needed to be set to either 0 or 1, the backward deterministic method is used to determine the logic values of these unspecified bits and justify the new specific states.

Cube intersection on the ON and OFF sets of the primary outputs or the next state lines is used if some unspecified bits are to be set to some specific logic values 1 or 0. The fault can then be propagated to the primary outputs or the next state lines. If this setting causes conflict in the unspecified bits between the true and faulty states, the differentiation sequence doesn't exist. Otherwise, when the unspecified bits are on the primary inputs, we just set them to the required values. When the unspecified bits are on the present state lines, we check if the present state is justified from the next state of the last clock cycle. If the justification step needs to set some unspecified bits in the present state lines of the last clock cycle to specific logic value 1 or 0, the same process is repeated on the last clock cycle. The new method will be discussed in detail in Section 7.2.

4.2.5 Fault Simulation

The quality of a test is measured or quantified by means of fault simulation. When a potential test sequence for a fault in sequential circuits is found, we fault simulate the sequence to check if it detects this fault and other faults. The sequence is first fault simulated by applying it to circuit models which have been altered slightly to imitate the effects of faults. If the circuit output response, as determined by simulation, differs from the response of the circuit model without the fault, then the fault is detectable by the sequence. After the process has been performed for a sufficient number of faults, an estimate

$$T = (\text{no. of faults detected})/(\text{no. of faults simulated})$$

is obtained which reflects the quality of the test sequence.

The fault simulation serves other purposes besides evaluating the test sequence [24]; in this thesis it:

- confirms detection of a fault for which an automatic test pattern generator (ATPG) claims that a successful test was found.
- computes fault coverage for a given test sequence.

Fault simulation is an important step in any ATPG system for both combinational and sequential circuits. Up to now, some efficient fault simulation algorithms have been developed. In general, there are three kinds of fault simulation methods, i.e., parallel fault simulation, deductive fault simulation, and concurrent simulation. In sequential circuits, the fault appears in every clock cycle. Hence, the single fault model becomes a multi-fault model.

In our system, we use a simple event-driven fault simulation. The algorithm of fault simulation is shown in Figure 4.5.

4.2.6 Determination of Redundant Faults

The difficulty in test generation for sequential circuits lies not only in testing difficult but testable faults, but also in the determination of redundant faults. Low fault coverage on certain circuits does not mean that a test generation system for sequential circuits is not suitable for the sequential circuit if we can show that the detected faults are close to the maximum possible number of detectable faults. In general, the determination of a redundant fault may need an astronomical amount of CPU time, because we should exhaust all the search space before the fault is considered as redundant.

There are two kinds of redundant faults in sequential circuits – combinational redundant and sequential redundant. For combinational redundant faults, it is relatively easy to detect them by using test generation for combinational circuits. The sequential redundant faults can be divided into two kinds: unjustifiable faults and undifferentiable faults [11]. If none of the excitation states are justifiable for a fault, the fault is said to be unjustifiably redundant. If there is at least a justifiable excitation state, but none of the excitation states have a differentiation sequence for a fault, the fault is said to be undifferentiably redundant.

The sequentially redundant faults can be found using theorem 1 in [21]. The theorem states that if all excitation states are not reachable from the reset state in the fault-free machine, the fault is sequentially redundant. We use the theorem for the

```
Input   : A sequence of test vectors and a fault under test.
Output : The fault is detected by the sequence or not.

Procedure Multi_fault_simulator(a fault under test) {
   for each clock cycle of test vector {
      deduct signals values at the unfaulted circuit;
      deduct signals values at the faulted circuit;
      for each primary output {
         if the unfaulted value is different from the faulted value
            return that the fault is detected by the test sequence;
      }
   }
   return that the fault can not be detected by the test sequence;
}
```

Figure 4.5. Fault simulation algorithm.

detection of sequentially-redundant faults as in [11]. We generate all combinational excitation states for a fault. If all excitation states are unjustifiable, the fault under test is redundant. A state is said to be unjustifiable if the number of fan-in cubes determined in state justification is zero or if all the fan-in states of the state are unjustifiable.

4.2.7 An Efficient Test Generation Algorithm for Sequential Circuits

The flow chart of the sequential circuit test generation algorithm based on the ideas presented above is given in Figures 4.6. As a preprocess, the algorithm starts with the extraction of the ON/OFF sets of the primary outputs and next state lines. For each fault under test, the sequential circuit test generation algorithm is given in Figures 4.7. The algorithm consists of:

Step 1. If the fault site is a node in the output cones of some primary outputs, the corresponding output cones of these primary outputs are extracted, and go to step 2. Otherwise, go to step 4.

Step 2. The transitive closure based test generation algorithm for pseudo-combinational circuits is used to find a (new) combinational excitation vector. If the combinational excitation vector has the present state part disjointed from the present state part of all the previously generated test vectors, go to step 3 to do state justification. If no such a new vector is found, the fault can't be propagated to the primary outputs directly, and go to step 4.

Step 3. State justification is used to find if the excitation state is reachable from the reset state. If the justification sequence is not found, return to step 2. If found, go forward to step 7.

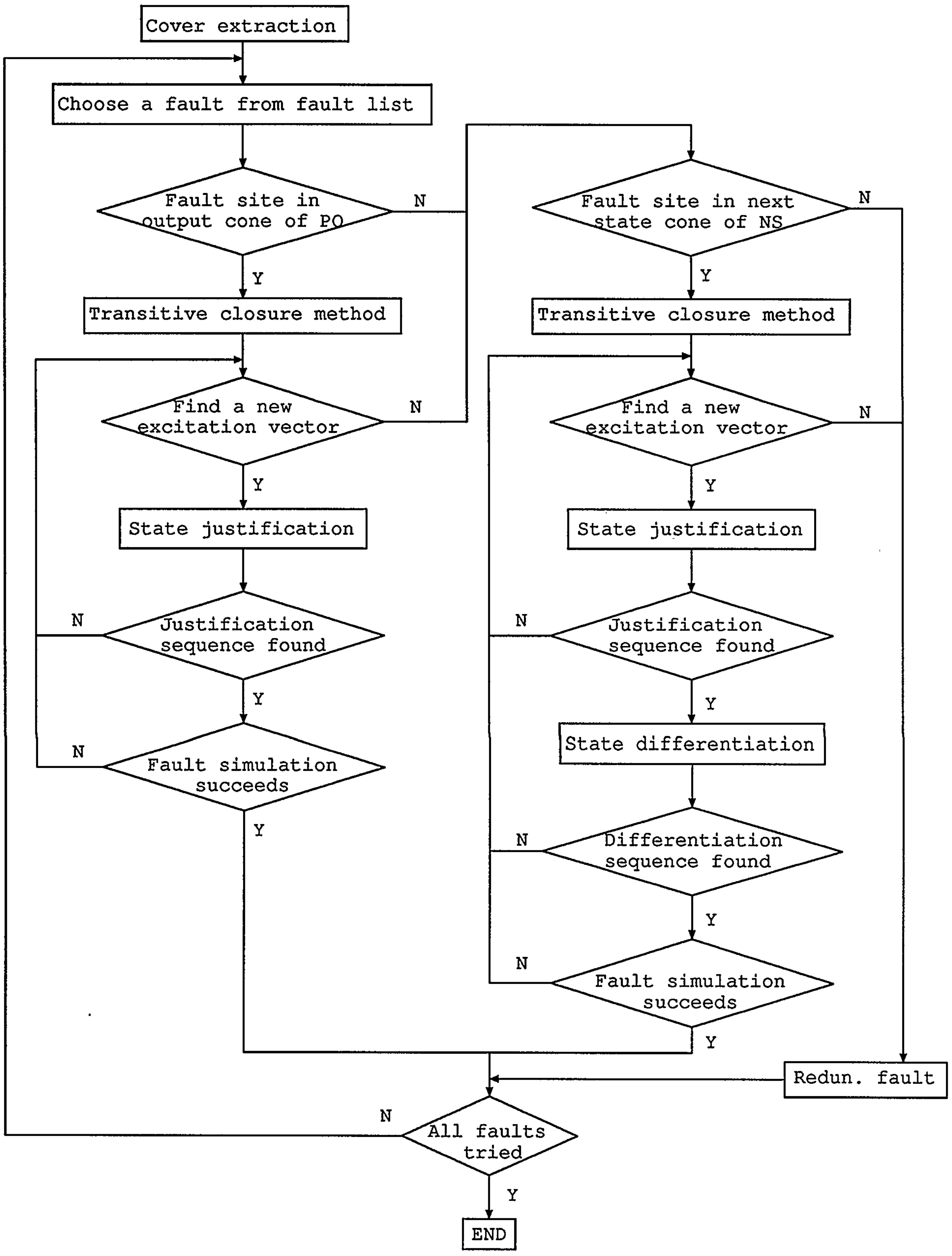

Cover extraction
Choose a fault from fault list
Fault site in output cone of PO
N
Y
Transitive closure method
Find a new excitation vector
N
Y
State justification
Justification sequence found
N
Y
Fault simulation succeeds
N
Y
Fault site in next state cone of NS
N
Y
Transitive closure method
Find a new excitation vector
N
Y
State justification
Justification sequence found
N
Y
State differentiation
Differentiation sequence found
N
Y
Fault simulation succeeds
N
Y
Redun. fault
All faults tried
N
Y
END

Figure 4.6. The flow chart of the test generation algorithm for sequential circuits.

```
Input   : The sequential circuit and a faults to be tested.
Output : A test sequence for the fault if found.

Procedure Sequential_test_generator(a fault under test) {
    /* try to propagate the fault to primary outputs */
    if the fault site is a node in output cones of some primary outputs {
        extract the output cones of these primary outputs;
        while a new combinational excitation vector is found {
            if justification sequence is found {
                /* the fault is detected by the test sequence */
                use the test sequence to fault simulate the fault;
                if fault simulation succeeds
                    return the test sequence;
            }
        }
    }
    /* try to propagate the fault to next state lines as it can't be
            propagated to primary outputs directly */
    if the fault site is a node in next state cones of some next state lines {
        extract the next state cones of these next state lines;
        while a new combinational excitation vector is found {
            if justification sequence is found {
                if differentiation sequence is found {
                    /* the fault is detected by the test sequence */
                    use the test sequence to fault simulate the fault;
                    if fault simulation succeeds
                        return the test sequence;
                }
            }
        }
    }
    redundant_fault_detect();
    if the fault is redundant
        return the fault is redundant;
    else return the fault is aborted;
}
```

Figure 4.7. The algorithm of sequential circuit test generation.

Step 4. If the fault site is a node in the next state cones of some next state lines, the corresponding next state cones of these next state lines are extracted, and go to step 5. Otherwise, the fault is redundant and exit.

Step 5. The test generation algorithm for pseudo-combinational circuits is used to find a (new) combinational excitation vector. If the combinational excitation vector has the present state part disjointed from the present state part of all the previously generated test vectors, go to step 6 to do state justification and state differentiation. If no such a new vector is found, exit without a test.

Step 6. State justification is used to determine if the excitation state is reachable from the reset state. If a justification sequence is not found, return to step 5. If found, state differentiation is performed to propagate the effect of the fault to the primary outputs. If a differentiation sequence is found, go to step 7. Otherwise, go back to step 5.

Step 7. Fault simulate the test sequence. If it detects the fault, return with the test sequence. Otherwise, go back to the previous step.

When a test sequence is found, the test sequence is used in simulating all undecided faults in the fault list. All the faults that can be detected by the test sequence are removed from the fault list.

4.3 Summary

Observations that initiated our research work in test generation for sequential circuits have been introduced in this chapter. A new approach which extends the transitive closure algorithm to test generation for sequential circuits has been developed. The efficiency of our method stems largely from the integration of several new algorithms. Our approach involves extracting the ON/OFF sets of the primary outputs and next state lines by adopting a new backward assignment method. The transitive closure algorithm has been extended to perform fault excitation by disabling all flip-flops in the sequential circuits. A novel backward deterministic method for state differentiation is developed to make our approach more efficient.

CHAPTER 5

COVER EXTRACTION

In state justification, we find a justification sequence from the reset state to the excitation state that propagates a fault to the outputs of the pseudo-combinational circuit. In state differentiation, we also need to find a differentiation sequence from the excitation state to the final state that propagates the effect of the fault to the primary outputs of the sequential circuit. In order to perform these two operations, we adopt cube intersections on the complete or partial ON/OFF sets of the primary outputs and next state lines. This process of extracting the ON/OFF sets of the primary outputs and next state lines is called cover extraction. In this chapter, we present in detail a new and efficient backward assignment method to perform cover extraction. This method has been successfully used by Yousif [30] to perform test generation for combinational circuits.

First, we present the backward assignment rules (referred to as B-rules). Then consistency and algorithm constraints are presented. Finally the new backward assignment procedure is presented. An example is used to illustrate the idea of the backward assignment algorithm.

5.1 Backward Assignment Rules (B-rules)

The objective of the B-rules is to propagate the assignment of a logic value at the output of a circuit to each node in the corresponding circuit cone during the backward assignment procedure. Figure 5.1 defines the B-rules used in the algorithm. A logic value is supposed to exist on a gate's output node, and the logic value assignments are carried out at the gate's input nodes.

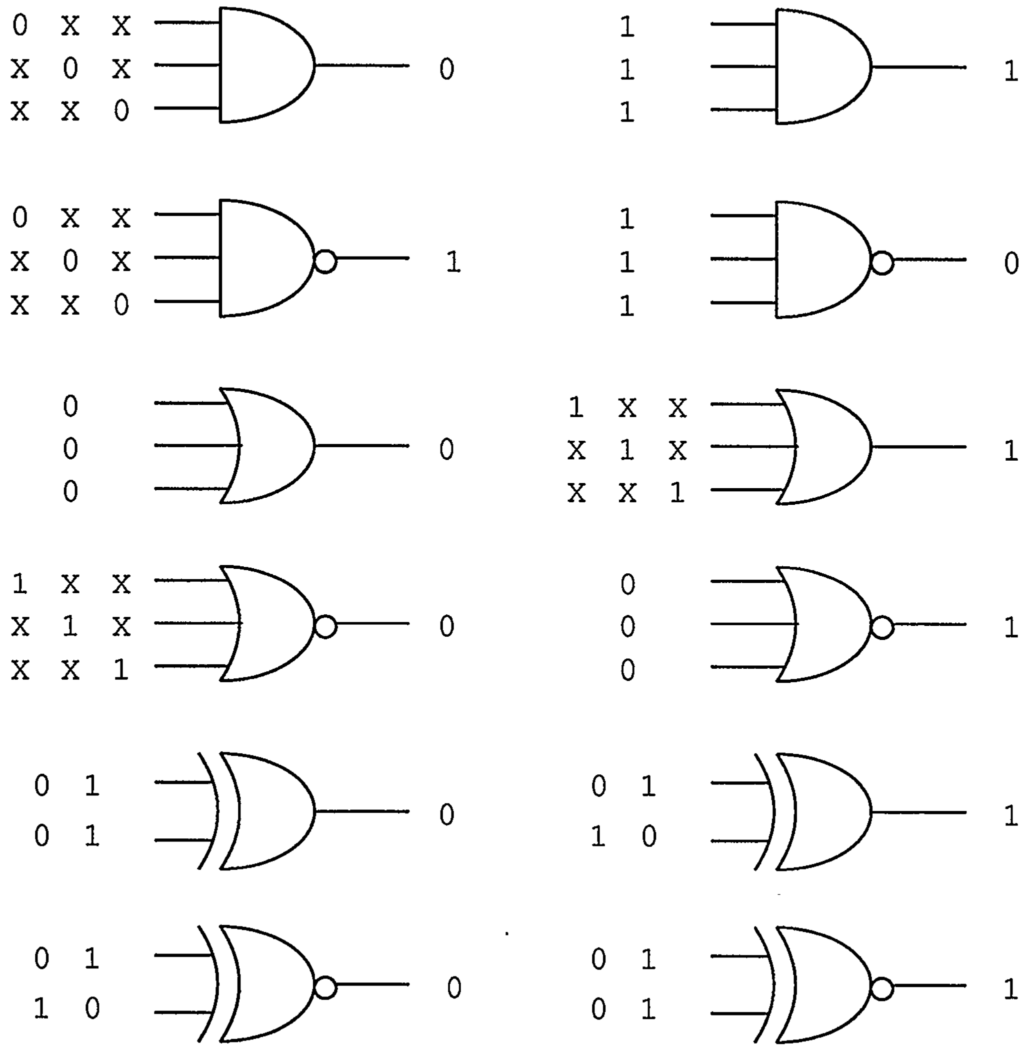


Figure 5.1. The backward assignment rules (B-rules).

For example, consider the 3-input AND gate shown in Figure 5.1. First, we want to find the OFF set of the output. From the K-map, we know that if any input of the AND gate is 0, the output is 0. So when the output is set to 0, one of the inputs must be 0. Three groups of inputs for the AND gate are obtained: $(0, X, X)$, $(X, 0, X)$, and

$(X, X, 0)$. Similarly, we can obtain the ON set of the 3-input AND gate, as shown in Figure 5.1, and one group of inputs is obtained: (1, 1, 1). If the output of the AND gate is X (don't care), all inputs are don't care. In this case, we just skip the output and leave the inputs to keep the original values.

5.2 Consistency and Algorithm Constraints

We use this backward assignment method to propagate logic values at the primary outputs and next state lines to the inputs of the circuit. As some nodes may have multi-fanout nodes, these nodes may be assigned logic values more than once by their multi-fanout nodes. Since different paths are not equal in length, some primary inputs or nodes may be assigned earlier than others. Therefore, it is necessary to check at each level of assignments for the primary inputs or nodes that have been assigned new values.

The consistency constraint is proposed to ensure that the assignments are correct, as shown in Figure 5.2. If these logic values are in conflict with each other, for example, one fanout requires the node to be logic value 0, and another fanout requires the node to be logic value 1, the assignment should be discarded. If logic value v_1 assigned by one fanout implies logic value v_2 assigned by another fanout, we should choose the consistent logic value v_1.

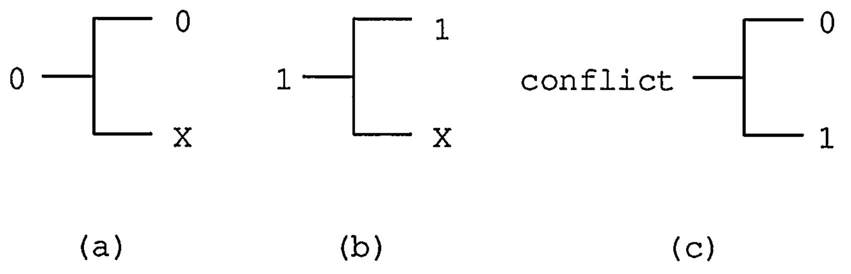


Figure 5.2. The consistency constraint.

The combination of the inputs may exceed one, for example, in the OFF set of the 3-input AND gate, three groups of inputs are obtained. In this case, every time, one group of assignments is used as the outputs of backward stage gates. Therefore, with the increase of the circuit's depth, the number of assignments for each node per level may increase dramatically. We use logic minimization to compress the assignments after the number of assignments reach a limit. This method assures that we obtain complete ON/OFF sets. Also it makes the storage of ON/OFF sets memory efficient.

5.3 The Backward Assignment Procedure

A high level description of the cover extraction algorithm is shown in Figure 5.3. For each output, the B-rules described earlier are used to extract the ON/OFF sets of the primary outputs and next state lines.

```
Input  : A sequential circuit's netlist.
Output : The ON/OFF sets of the primary outputs and next state lines.

Procedure cover_extract( ) {
  for each primary output and next state line {
    arrange the primary output or next state line in the list of
      node assignments;
    assign logic value 0 or 1 to it;
    while the list of node assignments is not empty {
      for each node in the list of node assignments {
        execute the backward-assignment function;
      }
      refresh the list of node assignments;
    }
  }
  return the ON/OFF sets of the primary outputs and next state lines;
}
```

Figure 5.3. Cover extraction algorithm.

To illustrate the idea of cover extraction, we use a simple sequential circuit s27 from the ISCAS'89 benchmarks shown in Figure 2.5. The algorithm *cover_extract* first selects an output node and assigns it logic value 0 or 1. Assume that *cover_extract* arbitrarily selects the next state line G11 and extracts the ON set (assign logic value 1). We need only consider the next state cone of G11, as shown in Figure 2.7. Node G11 now represents the only currently assigned node in the node list of the assignment and is assigned to logic value 1. According to the B-rules, both inputs of gate G11 are assigned logic value 0. At this point, a new level of assignment list includes two nodes G9 and G5. As node G5 is an input, its value is left unchanged and removed from the node list. From node G9, the algorithm assigns values $(1, X)$ and $(X, 1)$ to nodes G15 and G16. Nodes G15 and G16 are the two elements in the node list at this level of assignment. Starting at node G15 then node G16, *cover_extract* assigns logic values to nodes G12, G8 and G3. As node G8 is assigned twice, the consistency constraint is used. First, the first assignment $(1, X)$ for nodes G15 and G16 is considered. Node G15 requires nodes G12 and G8 to have the assignments $(1, X)$ and $(X, 1)$. As node G16 has logic value X, we can omit it and nodes G3 and G8 keep the logic value X. When these assignments are combined, nodes G12, G8 and G3 will have two groups of assignments $(1, X - X, X)$ and $(X, 1 - X, X)$. Here, $1 - X$ means that node G15 requires node G8 to be 1 and node G16 requires node G8 to be X, etc. According to the consistency constraint, we should set node G8 to logic value 1. The assignments for nodes G12, G8, and G3 are $(1, X, X)$ and $(X, 1, X)$. Similarly, from the second assignment $(X, 1)$ for nodes G15 and G16, the assignments for nodes G12, G8 and G3 are $(X, 1, X)$ and $(X, X, 1)$. So there are four assignments for node G12, G8 and G3: $(1, X, X)$, $(X, 1, X)$, $(X, 1, X)$, and $(X, X, 1)$. It is obvious that the second and the third assignments are the same. After logic minimization, only three groups of assignments $(1, X, X)$, $(X, 1, X)$, and $(X, X, 1)$ remains for nodes G12, G8 and G3. The process repeats until all values at the intermediate nodes are propagated to the inputs by using the B-rules. The final ON sets of G11 are $(X, 0, X, X, 0, X, 0)$, $(0, X, X, X, 0, 1, X)$, and $(X, X, X, 1, 0, X, X)$ for the inputs G0, G1, G2, G3, G5, G6 and G7. Figure 5.4 shows how the backward assignment procedure assigns logic values to the circuit nodes.

The obtained ON/OFF sets of the primary outputs and next state lines may be redundant. So after obtaining the ON/OFF sets, logic minimization is used to minimize the ON/OFF sets. The ON/OFF sets are represented as bit vectors which are similar to those used in ESPRESSO [4].

5.4 Summary

In this chapter, the backward assignment rules (B-rules) are presented. They propagate the assignment of logic value at the output of a pseudo-combinational circuit to each node in the corresponding circuit cone. The consistency constraint is proposed to ensure that the assignments are correct. Logic minimization is used to compress the ON/OFF sets.

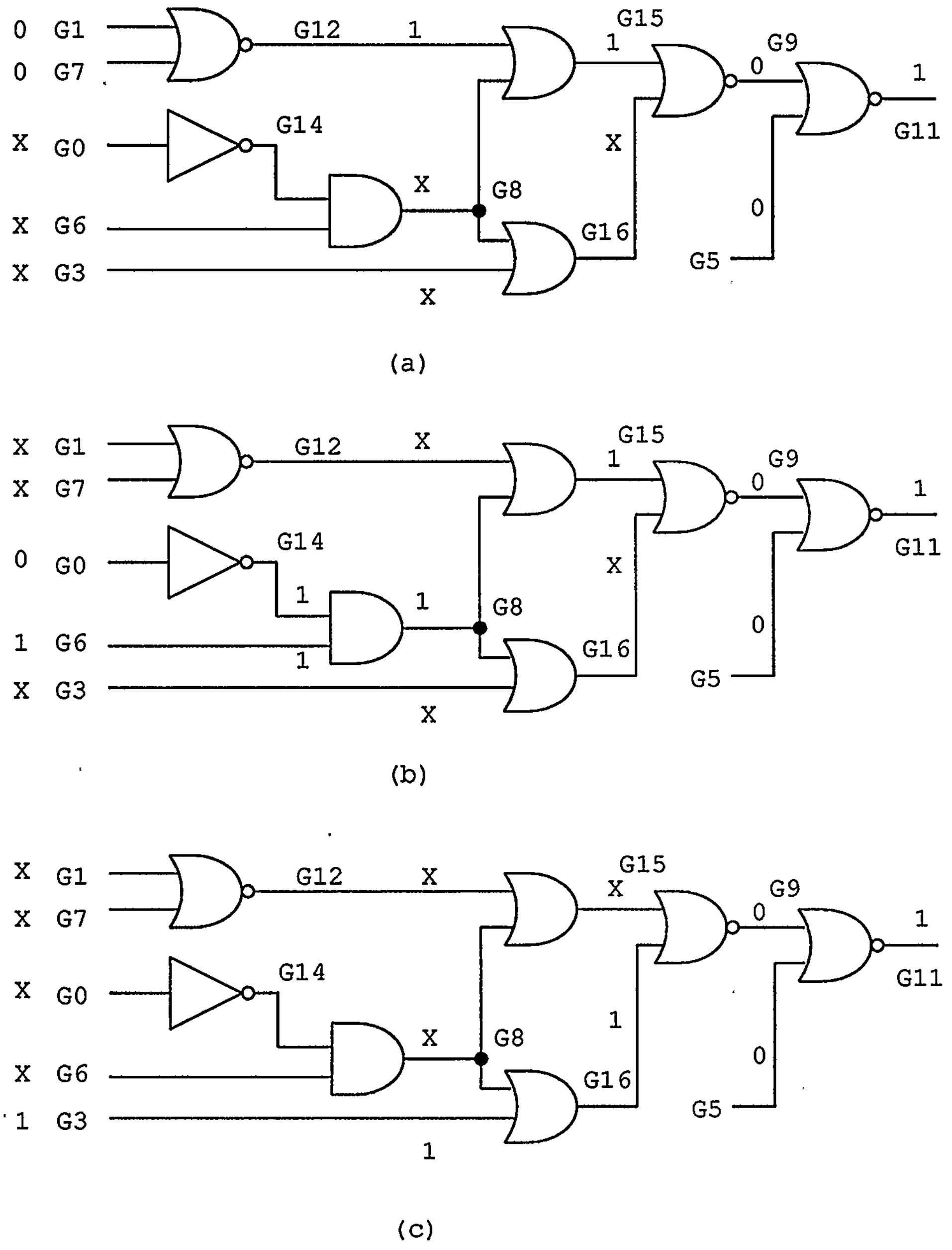


Figure 5.4. The next state cone of G11 in circuit S27 shows how the backward assignment procedure assigns logic values to the circuit nodes.

CHAPTER 6

PSEUDO-COMBINATIONAL CIRCUIT TEST GENERATION

To generate a test sequence for a fault in sequential circuits, we first generate a combinational test vector that propagates the effect of the fault to the primary outputs or the next state lines. In this chapter, we extract a formula that defines the structure of the related circuit and then use a transitive closure algorithm to satisfy the formula.

6.1 Circuit Representation

When all flip-flops in a sequential circuit are disabled, the sequential circuit becomes a pseudo-combinational circuit. The digital combinational circuit can be represented as a set of unary, binary, ternary, and M-ary ($M > 3$) relations.

6.1.1 Boolean Difference

In the 1960s and early 1970s, an algebraic or symbolic manipulation method called Boolean difference, differing from structural methods, appeared. This method did not achieve the popularity of the structural methods because of its complexity of computation. Since the test pattern generation using Boolean satisfiability was introduced in [19], this method has received more and more attention. First, the method of Boolean difference is described briefly.

Given a function $f(x) = f(x_1, x_2, ..., x_i, .., x_n)$ which describes the behavior of a combinational circuit, where $x_1, ..., x_n$ are the inputs of the circuit, we define the Boolean difference of $f(x)$ with respect to its ith input variable as

$$\frac{df}{dx_i} = f(x_1, ..., x_i, ..., x_n) \oplus f(x_1, ..., \overline{x_i}, ..., x_n) \tag{6.1}$$

Then

$$\left[X_i^a \cdot \frac{df}{dx_i}\right]_T = 1 \tag{6.2}$$

is the necessary and sufficient conditions of fault x_i stuck at a detected by vector T, where $a = 1$ or 0, $X_i^1 = \overline{X_i}$, and $X_i^0 = X_i$. Equation 6.2 implies that the fault under test is first excited to the logic value opposite to the stuck-at value, and then the change of the logic value at the fault location can be observed at the primary outputs. In short, test generation can be viewed as a search of an n-dimensional 0-1 space defined by the variables x_i $(1 \leq i \leq n)$ for points that satisfy the above equation.

6.1.2 Transferring Circuit into CNF

At first, the circuit is represented as the conjunctive normal form, i.e., CNF (also known as product of sums). As an example, a two input AND gate shown in Figure 6.1 is used to illustrate how to get CNF formula from a circuit.

The formula of the AND gate is

$$Z = X \cdot Y \tag{6.3}$$

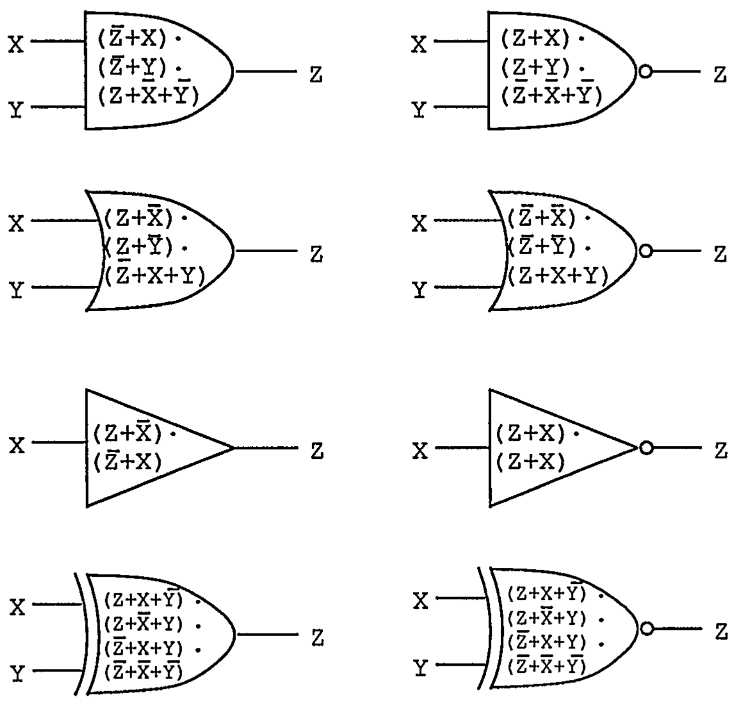

X
Y
$(\bar{Z}+X)\cdot(\bar{Z}+Y)\cdot(Z+\bar{X}+\bar{Y})$
Z
X
Y
$(Z+X)\cdot(Z+Y)\cdot(\bar{Z}+\bar{X}+\bar{Y})$
Z
X
Y
$(Z+\bar{X})\cdot(Z+\bar{Y})\cdot(\bar{Z}+X+Y)$
Z
X
Y
$(\bar{Z}+\bar{X})\cdot(\bar{Z}+\bar{Y})\cdot(Z+X+Y)$
Z
X
$(Z+\bar{X})\cdot(\bar{Z}+X)$
Z
X
$(Z+X)\cdot(Z+X)$
Z
X
Y
$(Z+X+\bar{Y})\cdot(Z+\bar{X}+Y)\cdot(\bar{Z}+X+Y)\cdot(\bar{Z}+\bar{X}+\bar{Y})$
Z
X
Y
$(Z+X+\bar{Y})\cdot(Z+\bar{X}+Y)\cdot(\bar{Z}+X+Y)\cdot(\bar{Z}+\bar{X}+\bar{Y})$
Z

Figure 6.1. The CNF formulae of basic gates.

and it is logically equivalent to the following CNF formula:

$$CNF = (\overline{Z} + X) \cdot (\overline{Z} + Y) \cdot (\overline{X} + \overline{Y} + Z) \tag{6.4}$$

It is obvious that if and only if the values of the variables are consistent with the truth table of the AND gate, Equation 6.4 equals to 1.

Figure 6.1 illustrates the CNF formulae for the basic gates (only one or two inputs). In the CNF formula, one sum is called a *clause* and each term in a clause is called a *variable*. Clauses with one, two, or three variables are unary, binary, or ternary clauses, respectively. It is convenient to extend the basic CNF formulae in Figure 6.1 to gates which have more than two inputs. For example, the CNF formula for a NAND gate with three inputs X, Y, and W is shown in Figure 6.2.

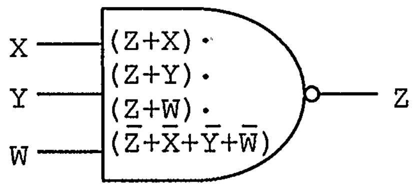


Figure 6.2. The CNF formula of 3-input NAND gate.

Considering the circuit example S1 shown in Figure 6.3. By extracting each formula for each gate in the circuit using the above method, the CNF formula for the output of the circuit is:

$$CNF = (D+\overline{A})\cdot(D+\overline{B})\cdot(\overline{D}+A+B)\cdot(E+C)\cdot(\overline{E}+\overline{C})\cdot(F+D)\cdot(F+E)\cdot(\overline{F}+\overline{D}+\overline{E}) \tag{6.5}$$

We will derive a test for the fault D s-a-0. The faulted circuit is produced by

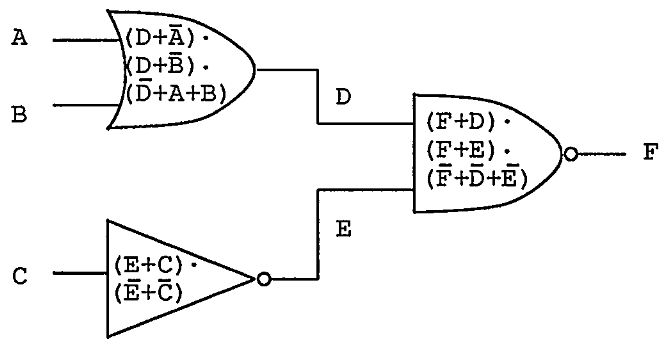


Figure 6.3. Formula extraction of a simple circuit S1.

copying the original circuit, renaming all related variables, and disconnecting the faulted site (all faulted signals are labeled with "′"), as shown in Figure 6.4. Because of the fault D s-a-0, the signal D.is always at logic value 0 no matter what values are at the inputs A and B. We disjoint the signal D to two signals: unfaulted D and faulted D'. In order to detect the fault, D' has logic value of s-a-0 and D must have logic value 1.

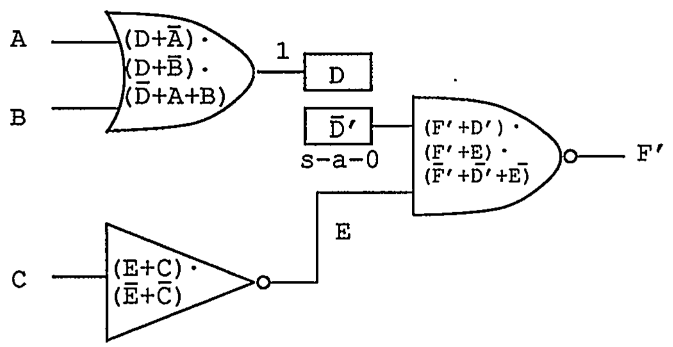


Figure 6.4. Formula extraction of the simple circuit S1 with a fault.

As the unfaulted and faulted circuits have the same behavior except those nodes that are affected by the fault, only the nodes that lies on a path between the fault site and a circuit output need to be renamed. The CNF formula for the faulted circuit is

$$CNF = (E + C) \cdot (\overline{E} + \overline{C}) \cdot (\overline{D'}) \cdot (F' + D') \cdot (F' + E) \cdot (\overline{F'} + \overline{D'} + \overline{E}) \tag{6.6}$$

It is not necessary to include the OR gate D in the CNF formula for the faulted circuit because of the implied discontinuity at the fault site.

According to Boolean difference, in order to detect the fault at D, the unfaulted and faulted circuits are put together and an XOR gate is added to their outputs. The final circuit is shown in Figure 6.5. BD is the output of the XOR gate. For the fault D s-a-0 to be covered, the output of the XOR gate should be 1. If the CNF formula equals to 1, a solution is found. Otherwise, no test exists. The formula of the final circuit is:

$$\begin{aligned} CNF = {} & (D + \overline{A}) \cdot (D + \overline{B}) \cdot (\overline{D} + A + B) \cdot (E + C) \cdot (\overline{E} + \overline{C}) \cdot (F + D) \cdot \\ & (F + E) \cdot (\overline{F} + \overline{D} + \overline{E}) \cdot (\overline{D'}) \cdot (F' + D') \cdot (F' + E) \cdot (\overline{F'} + \overline{D'} + \overline{E}) \cdot (BD) \cdot \\ & (\overline{F} + F' + BD) \cdot (F + \overline{F'} + BD) \cdot (\overline{F} + \overline{F'} + \overline{BD}) \cdot (F + F' + \overline{BD}) \end{aligned} \tag{6.7}$$

The problem of combinational circuit test generation can now be formulated as one of finding a consistent signal logic assignment which satisfies the above formula. The transitive closure method which is used to solve the problem will be presented in the next subsection.

6.1.3 Transitive Closure Method

On the basis of the CNF formula of the circuit, the transitive closure of the circuit is obtained in this section. As an example, consider the AND gate shown in Figure 6.1. Its CNF formula is given in Equation 6.4. We can transform the relationship

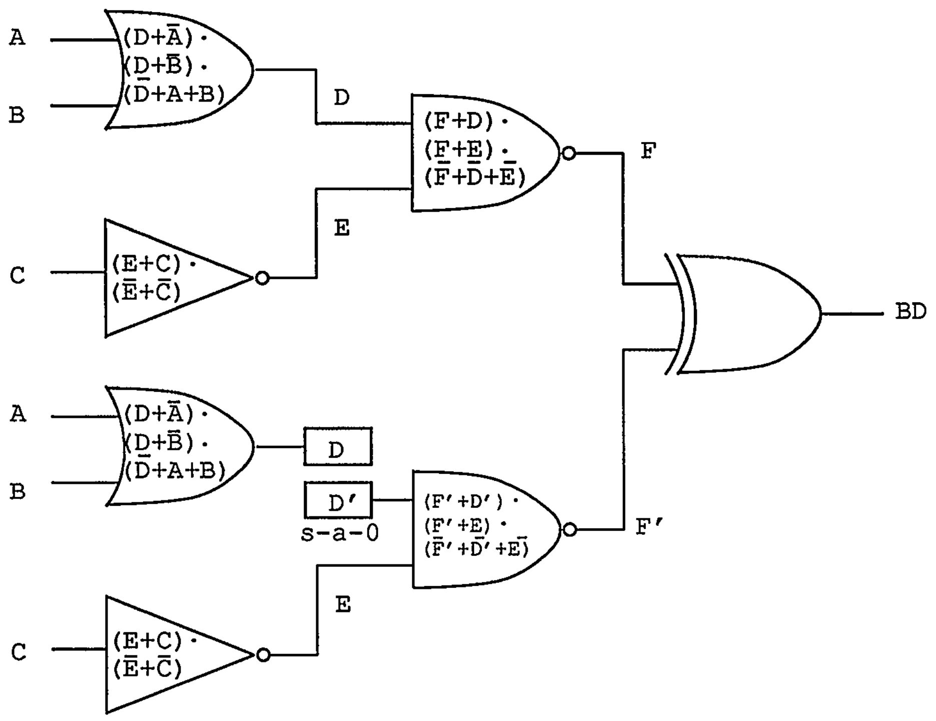


Figure 6.5. The XOR of the unfaulted and faulted circuits should be 1.

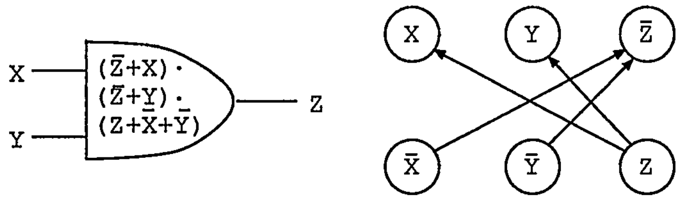


Figure 6.6. Implication graph of an AND gate.

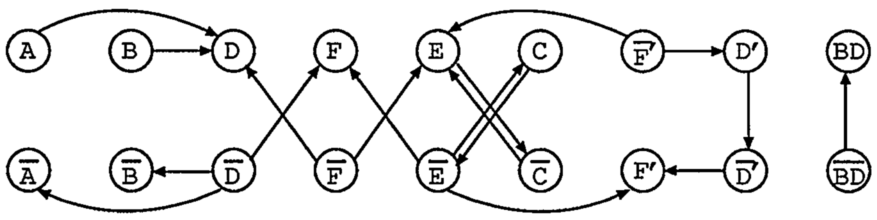


Figure 6.7. Implication graph of the example circuit S1.

into an implication graph, as shown in Figure 6.6. Where $\rightarrow$ donates implication. $\overline{X} \rightarrow \overline{Z}$ means that if $X = 0$, $Z = 0$, etc. This is consistent with the truth table of the AND gate. When $X = 0$, or $Y = 0$, Z must be zero. When $Z = 1$, X and Y must be 1.

In fact, the implication graph can be obtained from unary and binary clauses. For example, there are two binary clauses $(\overline{Z} + X)$ and $(\overline{Z} + Y)$ in the AND gate. When the formula is satisfied, each clause should be satisfied (equal to 1). For instance, to meet the clause $(\overline{Z} + X)$, when $X = 0$, Z should be 0; when $Z = 1$, X should be 1. From this clause, two implications are obtained: $\overline{X} \rightarrow \overline{Z}$, and $Z \rightarrow X$. The ternary or M-ary clauses can not be transfered to the implication form. But if we know or assume the logic values of one or more variables in these clauses, these clauses become binary clauses.

The implication graph and transitive closure of the simple circuit S1 with the fault D s-a-0 in Figure 6.3 are shown in Figure 6.7 and Table 6.1. In Table 6.1, '1' indicates that there is an edge and '0' indicates no edge between two signal nodes. For example, there is a '1' at the row of A and the column of D, so there is an edge from A to D.

As we discussed above, the implication graph can only be used to express unary and binary clauses, so is transitive closure. But besides these unary and binary clauses,

Table 6.1. Transitive closure of the simple circuit S1.

	A	$\overline{A}$	B	$\overline{B}$	C	$\overline{C}$	D	$\overline{D}$	E	$\overline{E}$	F	$\overline{F}$	D'	$\overline{D'}$	F'	$\overline{F'}$	BD	$\overline{BD}$
A	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
$\overline{A}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
B	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
$\overline{B}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0
$\overline{C}$	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
D	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{D}$	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0
E	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{E}$	0	0	0	0	1	0	0	0	0	0	1	0	0	0	1	0	0	0
F	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{F}$	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0
D'	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
$\overline{D'}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
F'	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{F'}$	0	0	0	0	0	1	0	0	1	0	0	0	1	1	1	0	0	0
BD	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{BD}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

Table 6.2. SAT representation of ternary for the simple circuit S1.

A	B	C	D	E	F	D'	F'	BD
1	1		-1					
			-1	-1	-1			
				-1		-1	-1	
					-1		1	1
					1		-1	1
					-1		-1	-1
					1		1	-1

there are still some ternary and M-ary ($M > 3$) clauses in the CNF of circuit. We use the satisfiability (SAT) form to express and save these ternary and M-ary ($M > 3$), as shown in Table 6.2. Where, '1' indicates that this is a positive variable and '-1' negative variable. For example, the first row can be expressed as $(A + B + \overline{D})$.

Many algorithms and approaches [13, 14, 16, 15] have been developed to solve the satisfiability problems.

With the combination of Tables 6.1 and 6.2, we can describe the circuit S1 with the fault D s-a-0 completely.

6.2 Efficient Transitive Closure Computation

Efficient transitive closure computation is introduced in this section. Signal dependencies are derived, and several methods are used to prune the search tree.

6.2.1 Signal Dependencies

Two kinds of signal dependencies are used: fixation and contradiction. If a path $x \rightarrow \overline{x}$ is found in the implication graph, it implies that x should be 0. Similarly, if a path $\overline{x} \rightarrow x$ is found, x should be 1. If both paths $x \rightarrow \overline{x}$ and $\overline{x} \rightarrow x$ are present in the implication graph at the same time, the contradiction exists and no solution can be found. By using this method, we can find if a variable is set to a value or not. So instead of obtaining the transitive closure of the implication graph, we just try to determine if there is a path between a variable and its complement. Here we use a breadth-first search algorithm to find a path between a variable x and its complement $\overline{x}$. The algorithm of signal dependency computation is quite simple, as shown in Figure 6.8.

If a contradiction occurs in the signal dependence, it means that some variable(s)

```
Input  : The directed graph $G = (V, E)$ and the assignment array of signals.
Output : The signal dependencies of the graph.

Procedure **transitive_closure**( ) {
  **for** each variable $v$ and its complement $\overline{v}$ {
    **if** a path from $v$ to $\overline{v}$ is found {
      **if** $v$ is assigned to 1 {
        /* contradiction */
        **return** no solution;
      }
      **else** $v$ is assigned to 0;
    }
    **if** a path from $\overline{v}$ to $v$ is found {
      **if** $v$ is assigned to 0 {
        /* contradiction */
        **return** no solution;
      }
      **else** $v$ is assigned to 1;
    }
  }
  **return** the signal dependencies of the implication graph;
}
```

Figure 6.8. The algorithm of signal dependency computation.

must be simultaneously assigned to logic values 0 and 1. In this case, there is no solution for this variable assignment. If signal values, which have been determined, satisfy the Boolean equation, the solution is found. Otherwise, a partial set of signal values determined may reduce some of the ternary relations to binary relations. These new binary relations are included in the implication graph and new signal dependencies should be determined. The process continues until no ternary or M-ary relations reduces to binary relations.

6.2.2 Pruning the Search Tree

According to our experience, the more constraints the variables have, the smaller the search tree. This is because when some variables are assigned to logic value 0 or 1, their relations with other variables may help us to determine other unassigned variables' logic values easily.

If a fault can propagate to one or more outputs of the circuit, there must be at least one sensitized path (similar to D algorithm) from the fault site to the output. In this path, the unfaulted and faulted values must be different. Suppose that if we add an XOR gate whose inputs are the unfaulted and faulted values, the output of the XOR gate must be one. The concept is similar to the active line variables used by others [19]. Let X be the unfaulted value, X' the faulted value, and EX the output of XOR gate whose inputs are X and X', we obtain these two clauses in ternary relation $(\overline{EX} + X + X') \cdot (\overline{EX} + \overline{X} + \overline{X'})$. If this path is active, that means $EX = 1$, X and X' must be different.

If a sensitized variable A has a single output, the clause $(\overline{EX_A} + EX_X)$ is added, which means that if A is the sensitized variable, X is sensitized. Also, if the sensitized variable A have two outputs X and Y, then the clause $(\overline{EX_A} + EX_X + EX_Y)$ should

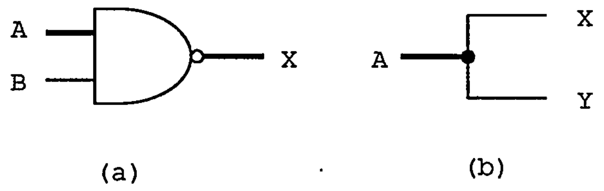


Figure 6.9. (a). If A is sensitized, X must be sensitized: $(\overline{EX_A} + EX_X)$. (b). If A is sensitized, either X or Y must be sensitized: $(\overline{EX_A} + EX_X + EX_Y)$.

be added. That means that if the variable A is sensitized, either the variable X or Y must be sensitized. Figure 6.9 shows two examples of these clauses.

On the other hand, some vertices in the directed graph may belong to a *strongly connected component.* So these vertices can be considered as one vertex. When the value of a vertex is obtained, the other vertices in this strongly connected component can be easily determined.

Because of the duality of the implication graph, if some vertices belong to a strongly connected component, the corresponding complemented vertices must belong to another strongly connected component. For example, in the circuit S1, there is a path from C to $\overline{E}$ and a path from $\overline{E}$ to C. Vertices C and $\overline{E}$ belong to a strongly connected component, and their complemented vertices $\overline{C}$ and E must belong to another strongly connected component. After transforming all implication relations with vertices C and $\overline{C}$ to vertices $\overline{E}$ and E, the vertices C and $\overline{C}$ can be deleted from the implication graph. By finding strongly connected components, the implication graph is condensed. The algorithm of finding strongly connected components in a directed graph can be found in [3].

Consider the simple circuit S1 with the fault D s-a-0 shown in Figure 6.4. In order to excite the fault, the logic values of D and D' must be different. The faulted value D' is 0, so the unfaulted value D should be 1. A clause (D) is added to the formula in Equation 6.7. D has one fanout F and an XOR gate has been added to the unfaulted line F and faulted line F'. After considering the sensitized path, the CNF formula is

$$CNF = (D+\overline{A})\cdot(D+\overline{B})\cdot(\overline{D}+A+B)\cdot(E+C)\cdot(\overline{E}+\overline{C})\cdot(F+D)\cdot$$
$$(F+E)\cdot(\overline{F}+\overline{D}+\overline{E})\cdot(\overline{D'})\cdot(F'+D')\cdot(F'+E)\cdot(\overline{F'}+\overline{D'}+\overline{E})\cdot(BD)\cdot$$
$$(\overline{F}+F'+BD)\cdot(F+\overline{F'}+BD)\cdot(\overline{F}+\overline{F'}+\overline{BD})\cdot(F+F'+\overline{BD})\cdot(D) \quad (6.8)$$

From the clause $(\overline{D'})$, we know that, in order to meet the CNF formula, D' must be set to 0. We can use the logic value of D' to simplify the CNF formula. As the clause $(\overline{F'}+\overline{D'}+\overline{E})$ is satisfied due to the logic value of D', we omit the clause. The clause $(F'+D')$ becomes (F'). From the new clause (F'), we know F' must be set to 1. We use the logic value of F' again to simplify the formula. The process continues until the formula can not be simplified any further. The final simplified CNF formula is:

$$CNF = (A+B)\cdot(E)\cdot(\overline{D'})\cdot(F')\cdot(\overline{F})\cdot(BD)\cdot(D) \quad (6.9)$$

There are no ternary clauses in the formula. The corresponding transitive closure is shown in Table 6.3.

6.3 Combinational Circuit Test Generation Algorithm

The test generation algorithm for combinational circuits based on the ideas presented above is as follows.

1. Derive the CNF representation of the combinational circuit with the fault. The

Table 6.3. Condensed transitive closure of the simple circuit S1.

	A	$\overline{A}$	B	$\overline{B}$	D	$\overline{D}$	E	$\overline{E}$	F	$\overline{F}$	D'	$\overline{D'}$	F'	$\overline{F'}$	BD	$\overline{BD}$
A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{A}$	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
B	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{B}$	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{D}$	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
E	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{E}$	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
F	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
$\overline{F}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D'	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
$\overline{D'}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
F'	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{F'}$	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0
BD	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{BD}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

unary and binary clauses are saved in the implication graph, and ternary and M-ary ($M > 3$) clauses are saved in the satisfiability form (SAT form).

2. Determine the transitive closure of the implication graph by using signal dependencies. If contradiction is found, the fault is combinational redundant. If variable values satisfy the Boolean equation, a solution is found without backtracking. Otherwise, only a partial set of variables are determined. These determined variables are used to reduce some ternary and M-ary ($M > 3$) clauses into binary clauses.

3. Find strongly connected components in the implication graph. A condensed implication graph is obtained.

4. Make a heuristic decision on the unassigned variables. Here we choose a decisive variable which is related to most ternary clauses.

5. If the pre-assigned backtrack limit is reached, we stop picking new variables, and go back to the previous decisive variables and assign the next logic value combination to them. Otherwise we pick the new variable determined in step 4 and assign the variable a value. The assignment may reduce some ternary relations to binary relations. These new binary relations are included into the implication graph.

6. If the variable assignments satisfy the Boolean equation, return the test vector. If a contradiction does not occur and the assignments satisfy a partial set of Boolean equation, we continue to do step 4. If a contradiction occurs, it means that the assignment does not satisfy the Boolean equation. Then we assign the decisive variable to the next value and repeat step 6. If there are no decisive variables left, we have implicitly exhausted the search space and no test vector is found.

The flow chart of the algorithm is shown in Figure 6.10.

For example, consider the transitive closure shown in Table 6.3. Two variables

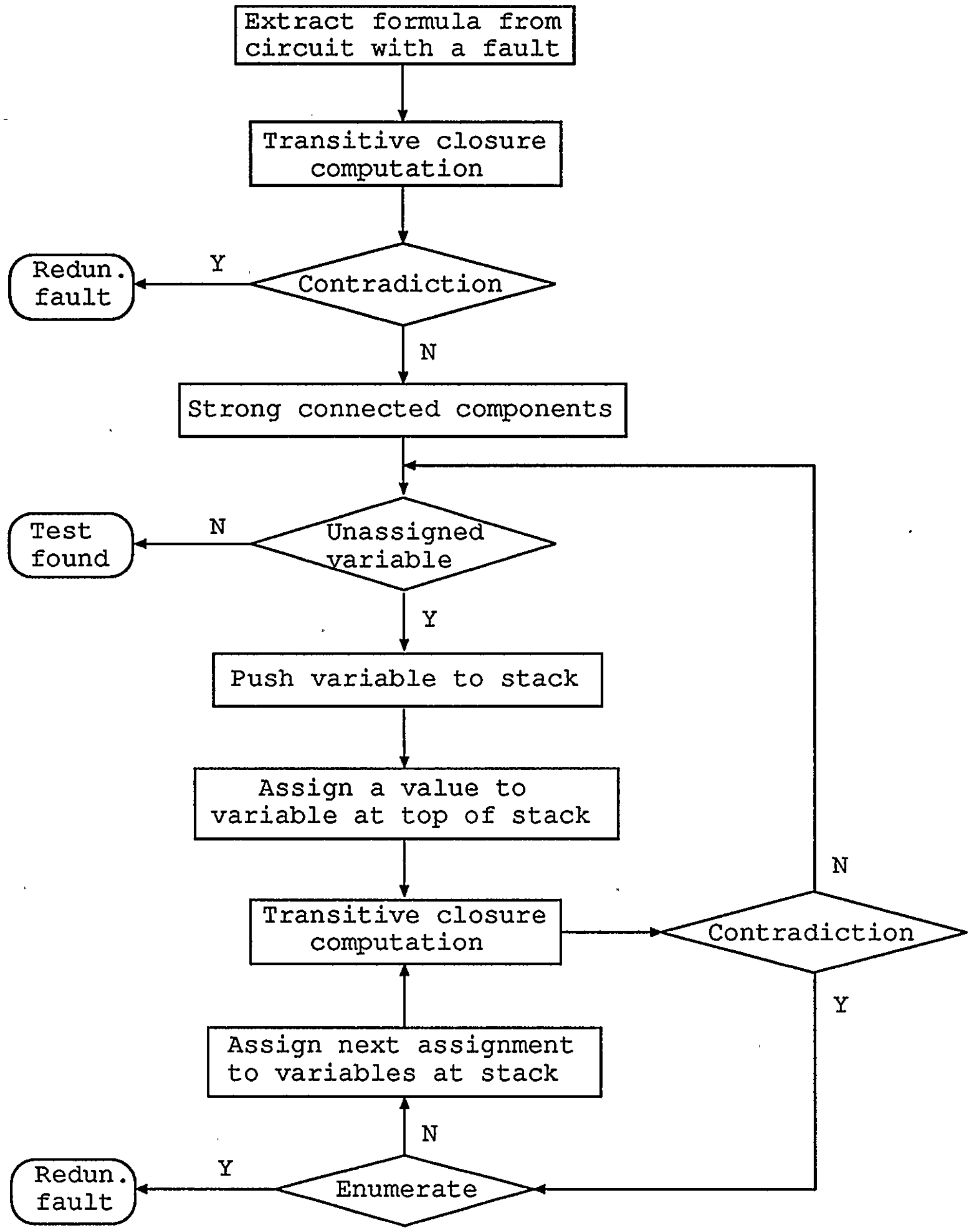

Extract formula from circuit with a fault
Transitive closure computation
Contradiction
Y
Redun. fault
N
Strong connected components
Unassigned variable
N
Test found
Y
Push variable to stack
Assign a value to variable at top of stack
Transitive closure computation
Contradiction
N
Y
Assign next assignment to variables at stack
N
Enumerate
Y
Redun. fault

Figure 6.10. The flow chart of the combinational test generation algorithm.

A and B are unknown and one clause $(A + B)$ is needed to be satisfied. Suppose that we choose A first, and set it to logic value 0. A path from A to $\overline{A}$ is added to the transitive closure. We derive new signal dependencies and find a path from $\overline{B}$ to B. So B should be at logic value 1 and the clause $(A + B)$ is satisfied. As C and $\overline{E}$ belong to a strongly connected component, and E has logic value 1, so C should be logic value 0. The whole CNF formula is satisfied, and we find that the vector $(0, 1, 0)$ for A, B, and C is an excitation vector for the fault D s-a-0.

6.4 Summary

The transitive closure method for generating test patterns for single stuck-at faults in combinational circuits is introduced in this chapter. It extracts a CNF formula from the model of circuit incorporating necessary conditions for fault activation and path sensitization, and then determines signal values which satisfy the formula. Several methods are used to prune the search tree. Instead of computing the entire transitive closure, we only concentrate on determining the signal dependencies of each variable and its complement.

CHAPTER 7

STATE JUSTIFICATION AND STATE DIFFERENTIATION

In three-phase ATPG, cover extraction is performed as a preprocess. The ON/OFF set information is stored in the bit's form. Test generation for sequential circuits is divided into three phases: combinational excitation vector generation, state justification, and state differentiation. In Chapter 5 and 6, we have described cover extraction and the pseudo-combinational test generation algorithm used in our system. Here we are going to describe state justification and state differentiation in detail.

7.1 State Justification

After an excitation state is found to propagate the fault to the primary outputs or the next state lines, state justification attempts to find a justification sequence from the reset state to the excitation state E_0. If the excitation state covers the reset state, the fault can be excited from the reset state, and state justification is not needed. Otherwise, state justification is used to justify the excitation state.

At first, the state justification algorithm tries to find a single-vector justification sequence from the reset state to the excitation state. The entire fan-in states E_1 can be obtained by cube intersections. The cubes of fan-in states are chosen according to the excitation state. If a present state line in the excitation state has logic value 1(0), the ON set (OFF set) of the corresponding next state line is picked. If a present state line has logic value X, the next state line is ignored and nothing is picked. The cube intersection of the ON and OFF sets of the next state lines gives the fan-in states of the excitation state E_0. The ON/OFF sets of the next state lines include both primary input and present state parts. The present state vectors are used to check if they cover the reset state and to get their fan-in states if needed. The primary input vectors are used to supply test sequence if the fault is detected. If the present states cover the reset state, the single-vector justification sequence is obtained.

If the single-vector state justification fails, we try to find a two-vector justification sequence. This is performed by attempting to justify the fan-in states E_1, via a single vector justification sequence. If the state justification algorithm succeeds, a two-vector justification sequence is found. Otherwise, a three-vector justification sequence is attempted. The process is repeated for the fan-in states of the state currently justified.

When we obtain the fan-in states, these states should be disjointed from the previously used states to prevent cycles. The state justification algorithm is shown in Figure 7.1. Figure 7.2 shows the algorithm of obtaining fan-in states of present state.

Consider the fault G2 s-a-0 in the circuit s27 shown in Figure 2.5. One of the excitation vectors is $(X, X, 1, X, X, X, 1)$ for G0, G1, G2, G3, G5, G6, and G7. So the corresponding excitation state is $(X, X, 1)$ for G5, G6, and G7. The states of G5 and G6 are the logic value X, so we can ignore them. As line G7 is at logic value 1, we pick up the ON set of G13. The ON set of G13 is $(X, 1, 0, X, X, X, X)$ and $(X, X, 0, X, X, X, 1)$ for G0, G1, G2, G3, G5, G6, and G7. From the first vector in the ON set, we know that when G1 and G2 have logic values $(1, 0)$, G13 has a logic value 1. In the next clock cycle, G7 would be logic value 1. As the reset state

```
Input   : The excitation state State and ON/OFF sets of next state lines.
Output : A justification sequence from reset to State if found; else
           return NOT-FOUND.

Procedure Justify_state(State) {
   /* put the primary input part of State into PI Stack */
   push State into PI Stack;
   get_fanins(State, Fanins);
   for each fan-in state Fanin in Fanins {
      if Fanin covers the reset state {
         return the state justification sequence saved in PI Stack;
      }
   }
   for each fan-in state Fanin in Fanins {
      Justify_state(Fanin);
      if the justification sequence is found {
         return the state justification sequence;
      }
   }
   Pop State from PI Stack();
   return (NOT-FOUND);
}
```

Figure 7.1. State justification algorithm.

```
Input  : The present state State and ON/OFF sets of next state lines.
Output : All fan-in states of State except those included in Exist-state
            (Used-state).

Procedure get_fanins(State, Fanins) {
   first_mark = TRUE;
   for each present state line that is a 1 or 0 {
      if first_mark is TRUE {
         Fanins = ON or OFF set of corresponding next state line;
         first_mark = FALSE;
      }
      else
         Fanins = Fanins ∪ (ON or OFF set of corresponding next state line)
   }
   /* do sharp produce to remove used cubes from Fanins */
   sharp_product(Fanins, Exist_state);
   /* logic minimization */
   minimization(Fanins);
   add_fanins_to_exist(Fanins);
}
```

Figure 7.2. The algorithm of obtaining fan-in states of present state.

$(0, 0, 0)$ implicates the states of G5, G6 and G7 (X, X, X) in the first vector of the ON set, the excitation state is reachable from the reset state. Table 7.1 gives the initial justification process. Where 0/1 means that 0 is the unfaulted value and 1 is the faulted value, etc.

Table 7.1. Initial state justification process.

	primary inputs				present states			next states			PO
gates	G0	G1	G2	G3	G5	G6	G7	G10	G11	G13	G17
justification vector	X	1	0/0	X	X	X	X	X	X	1/1	X
excitation vector	X	X	1/0	X	X	X	1	X	X	0/1	X

As the sequential circuit starts from the reset state, we should set the initial states of G5, G6 and G7 to $(0, 0, 0)$. After fault simulation, the justification sequence is a valid justification sequence. The final state justification is shown in Table 7.2.

Table 7.2. Final state justification process.

	primary inputs				present states			next states			PO
gate	G0	G1	G2	G3	G5	G6	G7	G10	G11	G13	G17
justification vector	X	1	0/0	X	0	0	0	X	X	1/1	X
excitation vector	X	X	1/0	X	X	X	1	X	X	0/1	X

7.2 State Differentiation

In our algorithm, the flip-flops are disabled and the sequential circuits are converted into pseudo-combinational circuits. For pseudo-combinational circuits, faults can be divided into three kinds:

1) The fault site is a node in the output cones of the primary outputs, and a combinational excitation vector can be found for the fault.

2) The fault site is not a node in any output cone, but is a node in the next state cones of the next state lines.

3) The fault site is a node in the output cones of the primary outputs, but a combinational excitation vector which propagates the fault to the primary outputs can not be found.

Consider the circuit shown in Figure 7.3. The fault $C\ s-a-0$ can propagate to the primary output K and belongs to the first kind of fault. For the fault $F \rightarrow I$ $s-a-0$, as I is not a node in output cone of primary output K, the fault belongs to the second kind of fault. For the fault $B\ s-a-1$, though B is a node in output cone of primary output K, the fault can not propagate to the primary output k directly. We should propagate the fault to the next state line I (the effect of the fault on I would propagate to the present state line J in the next clock cycle), and then propagate the effect of the fault on J to the primary output K. So the fault belongs to the third kind of fault.

For the latter two kinds, we should propagate these faults to the next state lines first by using the transitive closure based test generation method for combinational circuits. If a combinational excitation state is found and justified, state differentiation is used to propagate the effects of these faults on the next state lines to the primary outputs. If the stats differentiation algorithm succeeds, a differentiation sequence is found. Otherwise, the combinational excitation vector can not constitute a test sequence for the sequential circuit.

When the excitation vector propagates the fault to the next state lines, the true

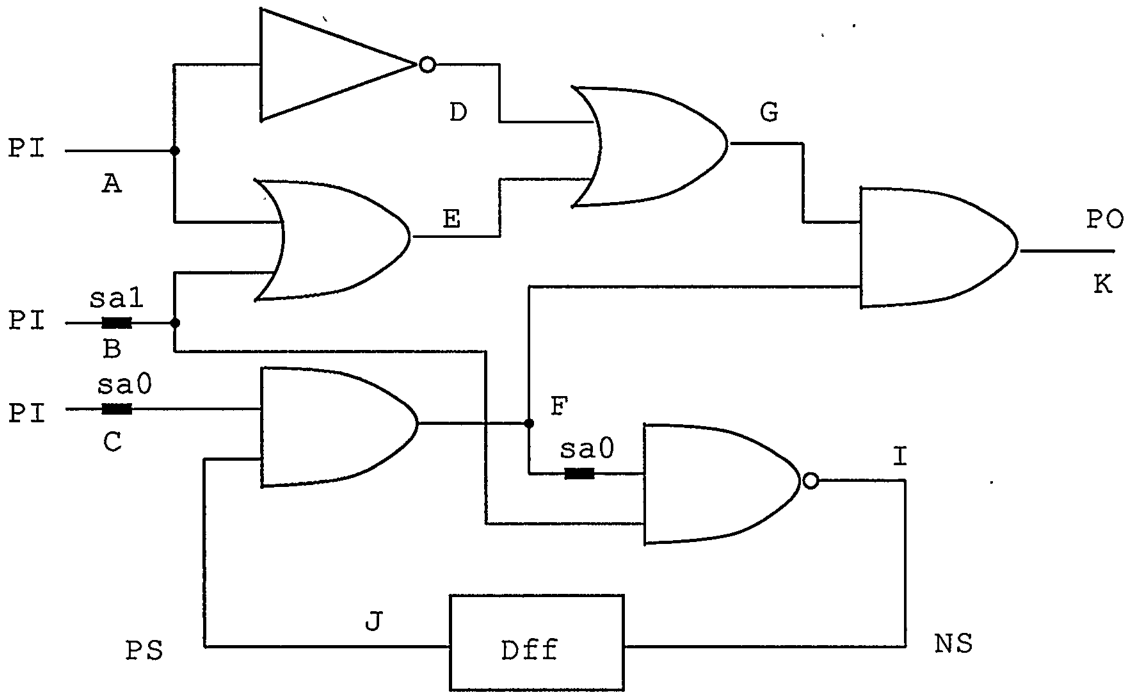


Figure 7.3. Three kinds of faults defined in our algorithm.

state S_1^T is the state in the fault-free circuit and the faulty state S_1^F is the state in the faulty circuit. S_1^T and S_1^F are guaranteed to differ in at least 1 bit. Since the effect of the fault has been propagated to S_1^F, we can assume that S_1^T and S_1^F are states in the fault-free circuit. The purpose of state differentiation is to find a differentiation sequence which causes S_1^T and S_1^F to have at least a different bit at the primary outputs.

To make the program more time-efficient, we use a random differentiation sequence as a first step. Some random vectors are added to the sequence starting from the reset state to the excitation state, and the unspecified primary inputs in the whole sequence are assigned random logic values. The whole sequence is used to fault-simulate the fault. If the sequence can detect the fault, then, a test sequence is found. Otherwise, a deterministic state differentiation method is used.

The idea of deterministic state differentiation is described as follows. According to the ON and OFF sets of every primary output, we search for a primary input vector which exists in both of the ON and OFF sets where the present state parts of the ON (or OFF) set and the OFF (or ON) set cover S_1^T and S_1^F separately. If such a primary input vector is found, the primary input vector constructs a single-vector differentiation sequence. The algorithm of single-vector state differentiation is shown in Figure 7.4. Otherwise, multi-vector differentiation sequences should be searched.

In multi-vector state differentiation, first, we try to find a two-vector differentiation sequence, then a three-vector sequence and so on. We attempt to propagate the true state S_1^T and the faulty state S_1^F to the next state lines by using a similar method as the one used in single-vector state differentiation. Instead of using the ON/OFF sets of the primary outputs in single-vector state differentiation, the ON/OFF sets of the next state lines are used. If the new true and faulty states are not found, quit

```
Input  : The true and faulty states $S^T$ and $S^F$, and the ON/OFF sets of
            primary outputs.
Output : A single-vector differentiation sequence if found; else
            return NOT-FOUND.

Procedure Single_vector_state_differ($S^T$, $S^F$) {
   for each primary output {
      /* find a primary input vector existed in the ON and OFF sets of
         the output */
      PI_vector = find_PI(ON-set, OFF-set);
      if PI_vector is found {
         /* judge if $S^T$ implies the ON-set and $S^F$ implies the OFF-set */
         Judge_implication($S^T$, ON-set, $S^F$, OFF-set);
         if implication is TRUE {
            return(PI_vector);
         }
         /* judge if $S^T$ implies the OFF-set and $S^F$ implies the ON-set */
         Judge_implication($S^T$, OFF-set, $S^F$, ON-set);
         if implication is TRUE {
            return(PI_vector);
         }
      }
   }
   return (NOT-FOUND);
}
```

Figure 7.4. The algorithm of single-vector state differentiation.

without solution. Otherwise, the single-vector state differentiation method is used to find single-vector differentiation sequence on the new true and faulty states (S_2^T, S_2^F) again. If found, a two-vector differentiation sequence is constructed. Otherwise, a three-vector differentiation sequence is attempted. The algorithm of multi-vector state differentiation is shown in Figure 7.5.

When the new true and faulty states are found, these states are disjointed from the used true and faulty states. Thus, cycles during state differentiation are prevented.

As test generation for combinational circuits produces an excitation vector with as many don't care entries in the primary inputs and present state lines as possible, if we just use the state differentiation algorithm described above, in most cases, we may not find a differentiation sequence even if it exists. This is because it is necessary to set the unspecified inputs in the test sequence to either 1 or 0. But some primary inputs and present states obtained by the transitive closure, state justification, and state differentiation may have some don't care entries. So in order to detect the fault, these don't care entries in the primary inputs and states have to be determined.

In STEED [11], all possible assignments to the unspecified inputs have to be made before it can be concluded that a test for the fault under consideration does not exist. There exists 2^n possible assignments for n unspecified inputs. Considering that each possible assignment may need to perform state justification, the real search space is much larger than 2^n.

We propose a new backward deterministic method for state differentiation. This method can help in finding the differentiation sequence and determining the don't care entries. When we attempt to propagate the fault to a primary output or next state line, if some present state lines in the current clock cycle are don't care entries,

```
Input  : The true and faulty states $S^T$ and $S^F$, and the ON/OFF sets of
           primary outputs and next state lines.
Output : A multi-vector differentiation sequence if found; else
           return NOT-FOUND.

Procedure Multi_state_differ($S^T$, $S^F$) {
    /* find all (new) excitation vectors fanouts which propagate $S^T$ and $S^F$
        to the next state lines */
    get_next_differ_state($S^T$, $S^F$, fanouts);
    for each fanout $i$ in fanouts{
        /* create new true and faulty states */
        create_new_states(fanout[i], $S_i^T$, $S_i^F$);
        /* use single state differentiation method */
        Single_vector_state_differ($S_i^T$, $S_i^F$);
        if found return (FOUND);
    }
    for each new true and faulty states $S_i^T$ and $S_i^F$ in fanout $i${
        Multi_state_differ($S_i^T$, $S_i^F$);
        if found return (FOUND);
    }
    return (NOT-FOUND);
}
```

Figure 7.5. The algorithm of multi-vector state differentiation.

while the same bits in the ON and OFF sets of the primary output or next state line are deterministic logic values, e.g., 0 or 1, we know that, in order to obtain a differentiation sequence, these present state lines must be set to the deterministic logic values. After we set these present state lines to the same deterministic logic values as in the ON and OFF sets, a new problem arises, i.e., whether the new specific present state is still justified from the previous clock cycles or not.

To solve the problem, in the backward deterministic method, we present a revised state justification algorithm to justify the specific present state. Because we have found a justification sequence from the reset state to the excitation state, in the revised state justification algorithm, we just need to specify some don't care entries in the justification sequence. When the specific present state requires that some of the don't care entries of the present state lines of the last clock cycle be set to deterministic logic values, the revised state justification algorithm is used again to justify the modified present state of the last clock cycle. The process continues until no more states are needed to be justified. If the specific present state is reached from the previous clock cycles, the state differentiation process continues. Otherwise, a differentiation sequence cannot be found for the true and faulty states.

When some don't care entries of the primary inputs are needed to be set to specific logic values, we just set them according to the ON/OFF sets and justification is not required. The backward deterministic algorithm for single-vector state differentiation is shown in Figure 7.6.

When the backward deterministic algorithm for single-vector state differentiation fails to find a single-vector differentiation sequence, we use a backward deterministic algorithm for multi-vector state differentiation. The algorithm is similar to the multi-vector state differentiation algorithm shown in Figure 7.5. The only two differences

```
Input   : The true and faulty states $S^T$ and $S^F$, and the ON/OFF sets of
            primary outputs.
Output : A single-vector differentiation sequence if found; else
            return NOT-FOUND.

Procedure Single_vector_back_state_differ($S^T$, $S^F$) {
  for each primary output {
    /* find a PI vector existed in the ON and OFF sets of the output */
    PI_vector = find_PI(ON-set, OFF-set);
    if PI_vector is found {
      /* get intersections: $S^T \cap$ ON-set, and $S^F \cap$ OFF-set */
      intersections($S^T$, ON-set, $S^F$, OFF-set);
      if both intersections are not empty {
        /* judge if some bits in $S^T$ and $S^F$ are X, while the same bits in
          both intersections are deterministic values */
        if some bits are needed to be set to specific values {
          /* set these bits to the specific values, and then justify the new
            deterministic state is reachable from the previous states */
          set_new_state();
          new_state_justification();
          if new state is reachable
            return(PI_vector);
        }
        else return PI_vector;
      }
      /* get intersections: $S^T \cap$ OFF-set, $S^F \cap$ ON-set */
      intersections($S^T$, OFF-set, $S^F$, ON-set);
      if both intersections are not empty {
        if some bits are needed to be set to specific values {
          set_new_state();
          new_state_justification();
          if new state is reachable
            return(PI_vector);
        }
        else return PI_vector;
      }
    }
  }
  return (NOT-FOUND);
}
```

Figure 7.6. The backward deterministic algorithm for single-vector state differentiation.

are:

- The backward deterministic algorithm for single-vector state differentiation shown in Figure 7.6 is used to replace the single-vector state differentiation algorithm shown in Figure 7.4.
- In order to propagate the fault to the next state lines, if some don't care bits in the state are needed to be set to specific values, we set them to the required values and then justify if the new specific state is reachable from the previous states.

To explain the idea of the backward deterministic state differentiation, we continue to consider the fault G2 s-a-0 in circuit s27 shown in Figure 2.5 as an example. The fault has been propagated to the next state line G13 and the justification sequence has been found. From Table 7.2, the true and faulty states are $(X, X, 0)$ and $(X, X, 1)$ for lines G5, G6 and G7.

First, we attempt to find a single-vector differentiation sequence. The ON and OFF sets of primary output G17 are shown in Table 7.3.

The intersection of the primary input parts on the first vectors of the ON set and the OFF set is not empty, i.e., $(1, 0, X, 0)$. The present state in the first vector of the ON set is $(X, X, 1)$, and is same with the faulty state. The present state lines in the first vector of the OFF set are $(0, X, 0)$, and its intersection with the true state $(X, X, 0)$ is not empty, i.e., $(0, X, 0)$. As the first bit in the true and faulty states is logic value X, in order to propagate the fault to the primary output, the first bit should be set to logic value 0. After setting, the differentiation vector becomes $(1, 0, X, 0, 0, X, 0)$. For the primary inputs, we just set them to the new logic values.

But the new differentiation state $(0, X, 0)$ should be justified if it is reachable from the previous clock cycle.

Table 7.3. The ON and OFF sets of primary output G17.

	primary inputs				present states		
gate	G0	G1	G2	G3	G5	G6	G7
ON-set	1	X	X	0	X	X	1
	X	X	X	0	X	0	1
	1	1	X	0	X	X	X
	X	1	X	0	X	0	X
	X	X	X	X	1	X	X
OFF-set	X	0	X	X	0	X	0
	0	X	X	X	0	1	X
	X	X	X	1	0	X	X

From the partial OFF set of the next state line G10, $(1, 1, X, 0, X, X, X)$, we know that if lines G0, G1 and G3 in the previous clock cycle are set to $(1, 1, 0)$, the new state would be reached from the previous clock cycle. The original vector in the previous clock cycle is the excitation vector, $(X, X, 1, X, X, X, 1)$, and its intersection with the OFF set of the next state line G10 is not empty, i.e., $(1, 1, 1, 0, X, X, 1)$. The excitation vector should be set according to the intersection. As all bits needed to be set are in the primary inputs, we just change the original logic value X to the new value. The single-vector differentiation sequence is found. The final test sequence is composed of the justification sequence, the excitation vector, and the differentiation sequence shown in Table 7.4.

After finding the test sequence we use it to fault simulate the fault G2 s-a-0, with results as shown in Table 7.4. The test sequence can detect the fault.

Table 7.4. The process of exciting the fault G2 s-a-0 to primary output G17.

	primary inputs				present states			next states			PO
gate	G0	G1	G2	G3	G5	G6	G7	G10	G11	G13	G17
justifi. vector	X	1	0/0	X	0	0	0	X	X	1/1	X
excitation vector	1	1	1/0	0	X	X	1	0	0	0/1	1/1
differen. vector	1	0	X/0	0	0	0	0/1	1/0	1/0	0/1	0/1

7.3 Summary

In this chapter, we have described two important steps in our ATPG system, state justification and state differentiation, in detail. State justification and state differentiation are efficiently performed using cube intersection on the ON/OFF sets of the primary outputs and next state lines. To increase the efficiency of the existing state differentiation in dealing with the unspecified inputs in the excitation vector and justification sequence, we have developed a new backward deterministic algorithm for state differentiation.

CHAPTER 8

EXPERIMENTAL RESULTS

8.1 IEEE Benchmarks

For an accurate evaluation of a test system, real circuit examples should be used. Benchmark circuits constitute a good example for evaluating a test system and also for comparing results with other systems. We have used the ISCAS'89 [6] sequential benchmark circuits to evaluate our test system. None of the ISCAS'89 benchmark examples have a specified reset state. We have assumed a vector of all zero to be the reset state, as in [11, 8]. Table 8.1 shows a subset of the ISCAS'89 benchmark circuits used in this research work. The five columns give the name and the numbers of primary inputs, primary outputs, flip-flops, and gates of each circuit.

As in some test generators for sequential circuits [11, 8], we have added a random fault simulator HOPE [20] as a front end to the deterministic test generation algorithm. Random vector test generation enables us to detect some of the easy to detect faults without much effort, and therefore, reduces test generation time. HOPE is an efficient sequential circuit parallel fault simulator which simulates 32 faults at a time.

8.2 Evaluation of the Proposed Test Pattern Generator

The test generation algorithm described in the previous chapters has been implemented in the program AST. It consists of about 10 000 lines of C code and runs in a UNIX environment. Table 8.2 gives the statistics of running AST for test generation.

Table 8.1. ISCAS'89 sequential benchmark circuit characteristics.

circuit	pi	po	dff	gate
s298	3	6	14	119
s344	9	11	15	160
s349	9	11	15	161
s382	3	6	21	158
s386	7	7	6	159
s400	3	6	21	162
s444	3	6	21	181
s510	19	7	6	211
s526	3	6	21	193
s526n	3	6	21	194
s641	35	24	19	379
s713	35	23	19	393
s820	18	19	5	289
s832	18	19	5	287
s953	16	23	29	418
s1196	14	14	18	529
s1238	14	14	18	510
s1423	17	5	74	657
s1488	8	19	6	653
s1494	8	19	6	647
s5378	35	49	179	2779

Experiments were performed on a SUN Sparc 10 workstation. For each circuit, the total number of faults (*total faults*), the number of detected faults (*det. fault*) and the number of faults that were proven redundant (*red. fault*) are given. The total fault coverage (*coverage*) includes detected and provably redundant faults. The next column reports the execution times in seconds. The total number of test vectors in test sequences is given in the column *test vec.*

Table 8.3 gives the statistical analysis of our system AST on the ISCAS'89 benchmark circuits. *RTG det. fault* is the number of faults detected by random test generation and *RTG time* is the time spent in random test generation. All columns under *AST* are the results obtained by our system AST. For each circuit, the number of faults detected (*det. fault*), the number of faults that were proven combinational redundant (*com. redun.*), the number of faults that were proven sequential redundant (*seq. redun.*), the number of faults that were aborted (*ab. fault*), and the execution times in seconds are given. The aborted faults are the number of faults aborted by the algorithm if it exceeds the backtrack limit set in the transitive closure method. The backtrack limit was set to 20 for all circuits except circuit s5378. For circuit s5378, the backtrack limit was raised to 50.

From Table 8.3, sequential random fault simulation was quite effective in generating tests. This may seem to contradict the accepted opinion that random sequences are ineffective for sequential circuits. Two reasons may explain the above results [8]:

- The availability of a reset state increases the effectiveness of random sequences.
- Sequential circuits in the ISCAS'89 benchmarks have fairly low sequence depth.

Figure 8.1 shows how CPU time is distributed among the procedures in AST. The transitive closure based test generation for combinational circuits(TC), state

Table 8.2. Real execution performance of our algorithm with the ISCAS'89 sequential benchmark circuits.

circuit	total faults	det. fault	red. fault	coverage (%)	time (sec)	test vec.
s298	308	273	35	100	2.4	192
s344	342	337	5	100	2.6	94
s349	332	325	7	100	2.7	95
s382	399	378	20	99.75	221	1431
s386	384	314	70	100	42	243
s400	424	396	27	99.76	1187	1382
s444	474	438	35	99.79	148	1247
s510	564	564	0	100	4.3	450
s526	555	462	89	99.28	593	2034
s526n	553	461	87	99.10	891	2105
s641	465	405	59	99.78	683	155
s713	581	480	101	100	341	248
s820	850	809	35	99.29	56	798
s832	870	812	51	99.20	45	818
s953	1079	1069	10	100	68	769
s1196	1242	1239	3	100	206	437
s1238	1355	1283	72	100	371	349
s1423	1515	1196	14	79.87	2653	4386
s1488	1486	1443	40	99.80	162	1069
s1494	1506	1455	51	100	238	1108
s5378	4603	3515	285	82.55	3745	1676

Table 8.3. Statistics analysis of our algorithm with the ISCAS'89 sequential benchmark circuits.

circuit	total faults	RTG det. fault	RTG time (sec)	AST det. fault	AST com. redun.	AST seq. redun.	AST ab. fault	AST time (sec)
s298	308	249	0.2	24	0	35	0	2.2
s344	342	315	0.2	22	0	5	0	2.4
s349	332	314	0.2	11	2	5	0	2.5
s382	399	257	1.5	121	0	20	1	219
s386	384	260	2.5	54	0	70	0	39
s400	424	307	1.9	89	6	21	1	1185
s444	474	402	2.2	36	14	21	1	146
s510	564	563	2.3	1	0	0	0	2
s526	555	392	3.0	70	1	88	4	590
s526n	553	356	3.0	105	0	87	5	888
s641	465	342	3.2	63	0	59	1	680
s713	581	433	3.2	47	38	63	0	338
s820	850	726	3.4	83	0	35	6	53
s832	870	732	3.3	80	14	37	7	42
s953	1079	964	3.8	105	0	10	0	64
s1196	1242	1112	3.9	127	0	3	0	202
s1238	1355	1137	4.1	146	69	3	0	367
s1423	1515	872	16.7	324	14	0	305	2636
s1488	1486	1353	4.4	90	0	40	3	158
s1494	1506	1381	4.8	74	12	39	0	233
s5378	4603	3156	22.4	359	40	245	803	3723

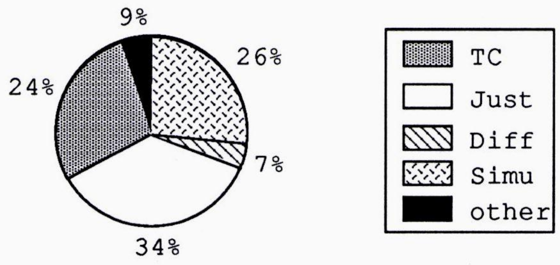


Figure 8.1. CPU time distribution.

justification(Just), state differentiation(Diff), and fault simulation(Simu) are listed.

Table 8.4 compares the results of AST to those of STEED [11] and VERITAS [8] on the ISCAS'89 benchmark circuits. The original CPU time of STEED was on a VAX-11/8800 and the CPU time of VERITAS was on DEC 5000/200. As the same in [8], we divided the times of STEED by 3 to obtain normalized DEC 5000/200 times shown in Table 8.4. The CPU time of AST was run on a SUN Sparc 10 station.

For most circuits, our system obtains more fault coverage than STEED with a smaller test sequence in a shorter CPU time. Considering the efficiency of STEED, the results of AST are mostly satisfying. For one particular circuit s5378, STEED obtained 99.3% fault coverage, and AST only got 82.55% coverage. This is due to the large number of flip-flops and the huge number of states, both STEED and AST fail to extract the complete covers. This adds to the difficulty of determining sequential redundant faults. STEED claimed that it obtained 30.25% redundant fault. Actually, STEED only found 69% detectable faults. But AST has found 76% detectable fault and only found 6.19% redundant faults.

When our system AST was nearly finished, we saw the results of VERITAS. VERITAS is faster, and gets 100% fault coverage in most circuits. But while STEED and our system AST produced results for $s1423$ and $s5378$, VERITAS ran out of memory with 80MB memory limit during its preprocess (reachability analysis). For our

Table 8.4. Test generation comparison with STEED and VERITAS on ISCAS'89 benchmark circuits.

circuit	coverage			time			test vectors		
	V	S	A	V	S	A	V	S	A
s298	100	99.0	100	4	5	2.4	119	280	192
s344	100	100	100	4	5	2.6	48	125	94
s349	100	100	100	4	5	2.7	56	120	95
s382	100	95.2	99.8	195	1320	221	1028	1633	1431
s386	100	100	100	3	4	42	168	238	243
s400	100	95.8	99.8	195	1200	1187	1091	409	1382
s444	100	95.6	99.8	152	1992	148	1026	994	1247
s510	100	99.8	100	7	7	4.3	584	733	450
s526	100	91.0	99.3	207	1060	593	1457	2037	2034
s526n	100	91.0	99.1	342	1040	891	1528	2287	2105
s641	100	93.1	99.8	15	10200	683	134	327	155
s713	100	93.1	100	21	10440	341	139	315	248
s820	100	100	99.3	40	120	56	785	1304	798
s832	100	99.7	99.2	49	120	45	763	1344	818
s953	100	100	100	40	29	68	578	1050	769
s1196	100	98.7	100	41	4080	206	376	545	437
s1238	100	99.0	100	52	3600	371	389	576	349
s1423	-	56.4	79.9	-	10800	2653	-	4026	4386
s1488	100	100	99.8	84	133	162	1031	1310	1069
s1494	100	100	100	103	147	238	1040	1374	1108
s5378	-	99.3	82.5	-	12000	3745	-	1037	1676
average		95.6	98.0		2776.5	553.3		1050.7	1004.1

system, if it can not extract complete covers due to the limit of memory and time, it could perform test generation on the partial covers, though it may fail to detect some faults or prove their redundancy.

From Tables 8.2 to 8.4, the performance of our algorithm can be evaluated as follows:

- The proposed algorithm outperforms the ATPG system STEED in terms of time complexity, fault coverage, and test length on most circuits of ISCAS'89 benchmarks.
- The fault coverage our system obtained is slightly lower than that of the new ATPG system VERITAS, but our system can perform test generation on large size sequential circuits.
- As the circuit size and number of flip-flops increase, the algorithm still shows an efficient performance. It has successfully generated tests for sequential circuits with a large number of flip-flops within reasonable amount of CPU time and has obtained close to maximum fault coverage.
- For some large circuits, when complete covers cannot be enumerated, the partial cover is generated and the algorithm can work on it.
- The proposed algorithm is useful as a deterministic algorithm for sequential circuit test generation.

8.3 Summary

In this chapter, the implementation of the system AST presented in Chapter 4 is discussed. Experimental results show that faults that require long test sequence are handled efficiently and finite state machines with a large number of states are tested using a reasonable amount of CPU time. Also our ATPG transitive closure based system can effectively determine a larger class of combinational and sequential redundant faults. Results show that considerable speedup factors and more fault coverages were realized due to the efficiency of the transitive closure algorithm and the powerful backward deterministic method for state differentiation. The overall test system yields a high fault coverage and provides time efficient procedures to generate tests for large size sequential circuits.

CHAPTER 9

CONCLUSIONS AND FUTURE WORK

The rapid advances in integrated circuit technology have made it possible to fabricate digital circuits with a very large number of devices on a single VLSI chip. The increase in size and complexity of circuits placed on a chip, with little or no increase in the number of input/output (I/O) pins, drastically reduce the controllability and observability of the logic on the chip. More logic must be accessed with almost the same number of I/O pins, making it much more difficult to test the chip. Yet, the need for testing is becoming more important. This research work proposes a new technique for designing test generation algorithms with better time complexity and fault coverage.

The test generation problem for sequential circuits has been presented as a state space search for test sequences which detect single stuck-at faults at the gate level of abstraction. It has been recognized that test generation for sequential circuits is a difficult problem. Different approaches have been used to tackle the test problem, either by randomly generating test sequences or by using deterministic test generation methods.

The current test generation algorithms for sequential circuits can generate test sequences for large sequential circuits. However, with increasing circuit complexity, either test generation time increases exponentially or it can not produce test sequences due to the exponential increase of reachable states. A new approach based on the transitive closure algorithm has been developed for the test generation of large sequential circuits. The similarities of this algorithm with current approaches have been identified.

As a preprocess, a new and efficient backward assignment method is presented to perform cover extraction. Logic minimization is used to assure that complete or maximum possible ON/OFF sets of the primary outputs and next state lines are extracted. By disabling flip-flops in the sequential circuits, the test generation problem for sequential circuits is transformed into test generation for combinational circuits. Then test generation for combinational circuits is formulated as the implication graph and the SAT form expressing necessary conditions for fault activation and path sensitization. A lot of techniques have been used to prune search trees. Our technique determines all logical consequences based on pairwise signal relationships for a partial set of signal assignments and provides a good framework for reasoning about signal relationships in the circuit.

After a combinational excitation vector is found, state justification is used to find a justification sequence from the reset state to the excitation state. If the effect of the fault is propagated only to the next state lines, state differentiation is needed to propagate the fault-effect to the primary outputs. To enhance the efficiency of state differentiation in dealing with the unspecified inputs in the test sequence, a novel backward deterministic algorithm for state differentiation is developed.

The implementation of the test generation algorithm for sequential circuits is presented with experimental results on the ISCAS'89 benchmark circuits. The results on large sequential circuits suggest that our algorithm outperforms the other test generation algorithms. Considerable speedup factors and more fault coverage are realized due to the efficiency of our test generation algorithm for sequential circuits.

The overall test system has yielded a high fault coverage and provided time efficient procedures to generate tests for large size sequential circuits. We have also shown that random patterns can be very effective in test generation for sequential circuit.

We believe that our algorithm can efficiently perform test generation for sequential circuits. It has obtained close to the maximum fault coverage on the ISCAS'89 benchmark circuits. Consequently, as was pointed out in [7], the parallelization of transitive closure computation, though not attempted in the present work, is easily possible. We hope that this system can be developed into parallel test generation systems.

REFERENCES

[1] V. D. Agarwal, S. K. Jain, and D. M. Singer. Automation in design for testability. In *Proc. Custom Integrated Circuit Conf.*, pages 21–23. Rochester, NY, May 1984.

[2] V. D. Agrawal, K.-T. Cheng, and P. Agrawal. Contest: A concurrent test generator for sequential circuits. In *Proc. 25nd Design Automat. Conf.*, pages 84–89, June 1988.

[3] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Reading, Addison-Wesley, MA, 1974.

[4] R. K. Brayton, G. D. Hachtel, Curt McMullen, and A. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis.* Kluwer Academic, Hingham, MA, 1984.

[5] M. A. Breuer and A. D. Friedman. *Diagnosis and Reliable Design of Digital Systems.* Computer Science, New York, 1976.

[6] F. Brglez, D. Bryan, and Kozminski. Combinational profiles of sequential benchmark circuits. In *Proc. IEEE Int. Symp. Circuits and Systems.*, pages 1929–1934, May 1989.

[7] S. T. Chakradhar, V. D. Agrawal, and S. G. Rothweiler. A transitive closure algorithm for test generation. *IEEE Trans. on CAD*, 12(7):1015–1027, July 1993.

[8] 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*, 12(7):935–945, July 1993.

[9] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[10] H. Fujiwara and T. Shimono. On the acceleration of test generation algorithms. *IEEE Trans. Comp.*, C-32:1137–1144, Dec. 1983.

[11] A. Ghosh, S. Devadas, and A. R. Newton. Test generation and verification for highly sequential circuits. *IEEE Trans. on CAD*, 10(5):652–667, May 1991.

[12] P. Goel. An implicit enumeration algorithm to generate tests for combinational logic circuits. *IEEE Trans. Comp.*, C-30:215–222, Mar. 1981.

[13] J. Gu. Efficient local search for very large-scale satisfiability problems. *SIGART Bulletin*, pages 8–12, 1992.

[14] J. Gu. Local search for satisfiability (SAT) problem. *IEEE Trans. on Systems, Man, and Cybernetics*, 23(4):1108–1129, Jul./Aug. 1993.

[15] J. Gu, X. Huang, and B. Du. A quantitative solution to constraint satisfaction. *New Generation Computing*, 13(1), Nov. 1994.

[16] J. Gu, P.W. Purdom, and B.W. Wah. Algorithms for satisfiability (SAT) problem: A survey. 1993. To appear.

[17] O. H. Ibarra and S. K. Sahni. Polynomially complete fault detection problems. *IEEE Trans. on Comp.*, C-24:680, March 1975.

[18] H. Kubo. A procedure for generating test sequences to detect sequential circuit failures. *NEC Res. and Dev.*, 12:69–78, Oct. 1968.

[19] T. Larrabee. Test pattern generation using Boolean satisfiability. *IEEE Trans. on CAD*, 11(1):4–15, Jan. 1992.

[20] H. K. Lee and D. S. Ha. Hope: An efficient parallel fault simulator for synchronous sequential circuits. In *29th ACM/IEEE Design Automation Conference*, pages 336–340, 1992.

[21] H. T. Ma, S. Devadas, A. R. Newton, and A. Sangiovanni-Vincentelli. Test generation for sequential circuits. *IEEE Trans. on CAD*, 7(10):1081–1093, Oct. 1988.

[22] S. Mallela and S. Wu. A sequential test generation system. In *Proc. Int. Test Conf.*, pages 57–61, Philadelphia, PA, Oct. 1985.

[23] A. Miczo. The sequential atpg: A theoretical limit. In *Proc. Int. Test Conf.*, pages 143–147, Oct. 1983.

[24] A. Miczo. *Digital Logic Testing and Simulation.* Harper and Row, Publishers, New York, 1986.

[25] P. Muth. A nine-valued circuit model for test generation. *IEEE Trans. Computers*, C-25:630–636, June 1976.

[26] J. P. Roth. Diagnosis of automata failures, a calculus and a method. *IBM J. Res. Dev.*, 10:278–291, July 1966.

[27] F. F. Sellers, M. Y. Hsiao, and L. W. Bearnson. Analyzing errors with Boolean difference. *IEEE Trans. Computers*, C-17:676–683, July 1968.

[28] S. Shteingart, A. W. Nagle, and J. Grason. Rtg: Automatic register level test generator. In *Proc. 22nd Design Automat. Conf.*, pages 803–807, Las Vegas, June 1985.

[29] J. J. Thomas. Automated diagnostic test program for digital networks. *Computer Design*, pages 63–67, Aug. 1971.

[30] A. S. Yousif. A novel search approach for test generation. Master's thesis, Dept. of Electrical and Computer Engineering, The University of Calgary, Calgary, AB T2N 1N4, September 1992.