THE UNIVERSITY OF CALGARY

A NOVEL SEARCH APPROACH FOR TEST GENERATION

by

Abdel-Fattah Sayed Yousif

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

September, 1992

ISBN 0-315-79111-X

Name Abdel-Fattah S. Yousif

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
SUBJECT TERM

0544
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.

SUJET

CODE DE SUJET

U·M·I

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
- Médiévale 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-être 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 pâturages 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 hôpitaux 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
- Généralités 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
- Électronique 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 systèmes 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 SEARCH APPROACH FOR TEST GENERATION", submitted by Abdel-Fattah Sayed Yousif in partial fulfillment of the requirements for the degree of Master of Science.

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

Dr. Jim Haslett
Dept. of Electrical & Computer Engineering

Dr. B. Nowrouzian
Dept. of Electrical & Computer Engineering

Dr. P. Kwok
Department of computer science

Date: Sept. 23rd, 1992

ABSTRACT

Advances in VLSI technology have now made it possible to integrate increasing number of devices on a single chip. The reliability of a chip is of foremost importance to a VLSI design engineer. Due to the increasing number of pins and the complex circuitry, it is very difficult to test a chip within an affordable cost.

Testing a chip involves generating a set of test vectors and their application to detect any faults in the circuit. Test generation is considered the most expensive step in the testing of a VLSI circuit. The test generation problem is known to be NP-complete, which in turn signifies the exponential time complexity of Automatic Test Pattern Generation (ATPG) algorithms. Although the test generation problem for combinational logic circuits is very well understood, it is difficult to achieve a significant breakthrough in reducing the time complexity of ATPG algorithms when testing large circuits.

The work presented in this thesis is a new approach for test generation of combinational logic circuits. In our algorithm, the test generation problem has been formulated as a global search problem which detects the sensitizing paths between the primary input and output nodes. In our approach, a tree of fault assignments is created in a similar manner to the depth-first search algorithms starting from primary outputs. The new approach avoids the time consuming backtracking procedure used in some other ATPG algorithms. The algorithm is designed to replace the traditional random test generators as a first phase in a test system. A model for a parallel test system is introduced which employs static load partitioning to equally balance loads

over the available processors.

The new test generation algorithm is tested using the ISCAS'85 benchmark combinational circuits. Our algorithm guarantees a minimum of 98% fault coverage of testable faults in all the benchmark circuits. The experimental results on large circuits indicate that our approach is much faster than any of the existing deterministic test generation algorithms. The experimental results with our parallel test model prove that a substantial efficiency improvement in speed can be obtained. The experimental results are compared with other existing test generation techniques.

Acknowledgement

I would like to express my gratitude and appreciation for the help and guidance given to me by Jun Gu throughout this research work.

I am also grateful to my colleagues R. Puri, H. Kenawy, and I. Williamson who were kind enough to read the contents of this thesis during its preparation and provide me with helpful comments.

Finally, I would like to acknowledge the financial support provided by the Electrical and Computer Engineering Department at the University of Calgary.

To

my family

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The advances in VLSI technology during the last decade have had a great impact on testing. Because of the increase in circuit size and the limited accessibility to the internal nodes of a circuit, the costs of testing a chip have become a substantial part of the overall chip costs. Most engineers would agree that the quality of an integrated circuit depends partly on the ability to test it. Testing now accounts for up to 10 percent of the total cost of manufacturing a 1-K RAM chip. For a 64-K RAM chip, the figure rises up to 40 percent. New techniques, however, promise help in the struggle to minimize costs, by tackling the circuit-testing problem in the design stage.

1.1 Overview of VLSI Testing

Testing is done in order to discover defects in a digital system. Test activities are interwoven with the VLSI design. Architectural design consists of partitioning a VLSI chip into realizable functional blocks. The logic design of these blocks should be synthesized in a testable form or the synthesized logic should be analyzed and improved for testability.

Faulty VLSI chips could be produced during manufacture because of photolithography errors, deficiencies in process quality, or improper design. Even if the chip is manufactured perfectly, it could subsequently wear-out in the field due to electromi-

Table 1.1. Types of tests performed.

Wafer	Chip	Board	System
Parametric	Chip test	Incoming inspection	System test
Die probe	Burn-in test	Board functional	Field diagnosis

gration, hot-electron injection, or other reasons. Environmental effects, such as alpha particles and cosmic radiation can also cause a circuit to produce erroneous data.

Testing is done at various stages in the production of a system: the dies are tested during fabrication, the packaged chips before insertion in the boards, the boards after assembly, and the entire system when complete. Table 1.1 indicates some of the tests performed at the various stages during the manufacturing process of a system.

As far as the level of VLSI chip testing is concerned, a test generation algorithm is used to provide the necessary test vectors which, if applied to the chip, will expose the faults occurring at this level of manufacturing . The test cost at this level is primarily determined by the cost of generating these test vectors. Consequently, a new discipline has emerged to probe the testability problem of a circuit more thoroughly in order to give the designer feedback without taking the risk of submitting a circuit design which is not testable. As some designers like to put it, a testable design is an optimal design. Indeed, design for testability has been very well recognized and served by many researchers.

When considering which test patterns to generate for testing a complex circuit, one should first consider how good the patterns are for detecting the possible physical failures in the circuit. It may be impossible to consider all possible physical failures. Hence, test patterns are generated to detect some set of modeled faults in the circuit. For example, any line in the gate-level representation of the circuit permanently stuck

at logic 0 or 1. The measure of test quality in this case could be the percentage of these stuck-at faults detected by the patterns, and is called *fault coverage* for the fault class. A typical goal might be to achieve a fault coverage for single stuck faults of 99% for the chip.

Fault coverage is determined by a fault simulation program. Simulation of all faults in a large circuit with many tens of thousands of gates may take a prohibitive amount of computer time. Statistical sampling procedures for simulating a fraction of the total faults are commonly used for measuring the effectiveness of the test patterns. It was found that for large circuits having a few thousand gates, reasonably good results can be obtained by simulating only 1000 to 2000 faults.

1.2 Testability Analysis

There are various factors that contribute to testing and its cost. Testing cost is determined mainly by the cost of test pattern generation and by the cost of 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 the test equipment plus the tester time required to apply the test. The cost of testing can be reduced by using tests which either fail to detect many faults or cannot locate many of the detected faults. This can cause a substantial increase in system production and maintainence costs. It is much more expensive to repair a faulty printed circuit board than to discard a faulty chip, and it is much more expensive to repair a faulty system than to repair a faulty printed circuit board. Consequently, more efforts were urgently needed to understand and formulate the test problem such that fast and efficient techniques for test pattern generation be provided. The main objective is not to compromise the quality of the test patterns, that is, the fault coverage should remain high.

Attempts to understand circuit attributes that influence testability have produced the two concepts of *observability* and *controllability*. Observability refers to the ease with which the state of internal signals can be determined at the circuit output leads. Controllability refers to the ease of producing a specific internal signal value by applying signals to the circuit input leads. Many of the Design For Testability (DFT) techniques are attempts to increase the observability or controllability of a circuit design. A straight forward approach to do this is to introduce test points, that is, additional circuit inputs and outputs to be used during testing. There is always a cost associated with adding test points. For circuit boards adding test points is often well justified. On the other hand, for ICs, the cost of test points can be prohibitive because of IC pin limitations.

A straightforward method for determining the testability of a circuit is to use an Automatic Test Pattern Generation (ATPG) program to generate the tests and determines the fault coverage. The running time of the program, the number of test patterns generated, and the fault coverage then provide a measure of the testability of the circuit. The difficulty with this approach is mainly the large expense involved in running the ATPG program. Also, the ATPG program may not provide sufficient information about how to improve the testability of a circuit with poor testability. To overcome these difficulties, a number of programs have been written to calculate estimates of the testability of a design without actually running an ATPG program such as TEMAS (Testability Measure Program) and SCOAP (Sandia Controllability/Observability Analysis Program) [9].

These Testability Measure (TM) programs implement algorithms that attempt to predict for a specific circuit the cost (running time) of generating test patterns. In the process of calculating the testability measure, information is developed identifying

those portions of the circuit which are difficult to test. This information can be used as a guide to circuit modifications that improve testability.

No accurate relationship between circuit characteristics and testability has yet been demonstrated. Thus the circuit parameters calculated by the TM programs are heuristic and have been chosen on the basis of experience and study of existing ATPG programs. It is not surprising that the various authors of TM programs have chosen different circuit characteristics for their estimates of testability. The technique used to demonstrate that a given TM program does indeed give an indication of circuit testability is to run both the TM program and also an ATPG program on a number of different circuits. A monotonic relation between the TM and the ATPG run time is offered as a "proof" that the TM program produces a good estimate of circuit testability. The difficulty with this validation technique is the high cost of running enough examples to be reliable. Some interesting results obtained by using statistical methods to evaluate the testability measure program approach are presented in [2].

1.3 Faults in VLSI Systems

As systems increase in complexity, it is useful to be able to describe faults at various levels of abstraction in the system. A fault which is described at a very low level, for example the level of transistors, may very accurately describe the physical phenomena causing the fault but, because of the extremely large number of transistors in a VLSI chip, the model may be intractable for the purpose of deriving tests for the fault. The two requirements for fault models are accuracy and tractability. Accuracy means realistic faults should be modeled, while tractability implies that very complex systems should be handled. These requirements are in some sense contradictory. Recent research, therefore, deals with deriving realistic models at higher levels which can accurately capture the faults at lower levels.

As an example, consider a contact between two conducting lines in a VLSI circuit. If the contact is faulty, then the fault can be described at this level of abstraction as a break between two lines. It may also turn out that the break is equivalent to the input of a gate being permanently set to logic 0. The fault can then be described at the gate level of abstraction as a stuck-at 0 fault. It would be simpler for the purpose of analysis to consider the fault at the highest possible level of abstraction.

Now suppose that contact is not permanently open but is periodically open, depending on the temperature, vibration, or other external causes. This can be described as an intermittent fault. However, testing for the fault may never expose it since the fault may not be active during the test. If, however, the circuit is designed so that the fault caused by the line being open can be detected during normal computation by using some error correcting techniques, the goal of dependable operation can still be achieved.

A physical failure can also lead to the output of a module being at a nonlogical value (for example, indeterminate level between logic 0 and 1). Such faults are difficult to describe and detect, but the errors due to these faults may also be detected by error detection techniques.

1.3.1 Fault Models

Fault models are descriptions of the effect of a defect or failure in a circuit. As discussed earlier, fault models are driven by the requirement to dèrive high quality tests for complex circuits. Thus a useful fault model will naturally lead to a test generation procedure for the fault.

1.3.1.1 Transistor-level Fault Models

Defects in present day integrated circuits can be abstracted to shorts and opens in the interconnects and degradation of devices. Fault models at the transistor level, therefore, can characterize physical failures quite accurately. Unfortunately, as the complexity of VLSI increases, the number of potential faults at the device and interconnect level also increase drastically. Nevertheless, it is necessary to study the effects of failures at the transistor level and to develop accurate fault models at this level. Better understanding of the effects of failures can be used to develop accurate fault models at higher levels which can be applied to complex systems. This approach is analogous to that used in the hierarchical design of VLSI systems where complex circuits are built from smaller cells.

Fault models proposed at the transistor level incorporate one or more of the following classes of faults:

- shorts and opens of transistors or interconnections.
- delay effects of failures.
- coupling or crosstalk between nodes of a circuit.
- degradation of elements.

Shorts and opens are included in most fault models while the more accurate and more complex models include delays. Fault models where activity on one node affects the logic values on another node in the circuit are primarily applied to memories. Fault models which incorporate degradations of elements (for example, transistor parameter changes, or changes in the value of a resistor) are usually used in analog circuits.

1.3.1.2 Gate-level Fault Models

Early fault models were developed at the logic gate level. The popularity of this approach can be attributed to several reasons.

- Such models are simple to design and use.
- Many faults in discrete technologies can be represented by faults at the logic gate level.
- Use of such fault models allows many of the powerful results in mathematics relating to Boolean algebra to be applied to deriving tests for complex systems.
- A fault model at the logic gate level can be used to represent faults in many different technologies if, in fact, defects and faults in these technologies can be mapped to gate faults.

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 0 or 1. This model has been the source of a great deal of research. It is still very popular since it has been shown that many defects at the transistor and circuit level can be modeled by the stuck-at fault model at the logic level. In practice, only single stuck faults are considered in a circuit.

A subset of the stuck fault model is the pin fault model, where only input/output pins of a module are assumed to be stuck-at 0 or 1 under failure. This has been used sometimes when testing printed circuit boards with many VLSI devices. Unfortunately, this fault model does not even include a high percentage of gate level stuck faults within the module in most cases and is, therefore, inappropriate for VLSI.

1.4 Fault Equivalence and Dominance

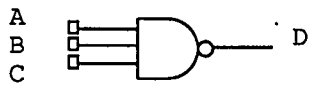

Figure 1.1. A three-input NAND gate example

Consider the three input NAND gate shown in Figure 1.1. This gate has four lines (three inputs and one output) and would, therefore, have eight stuck-at faults, each line stuck at 0 or 1. However, the faults A, B, or C stuck-at 0 would result in the output D being permanently 1 and, therefore, it is impossible to distinguish between an input stuck at 0 from the output stuck at 1. These faults are said to be *equivalent.* Now consider the fault A-stuck-at-1. In order to detect this fault, a 0 has to be applied on A, and 1s at B and C so that the effect of the fault can be propagated to D. The correct value of D will be a 1 and it will be a 0 under fault. This test for A-stuck-at-1 will, therefore, also detect the fault D-stuck-at-0. Hence, A-stuck-at-1 is said to *dominate* D-stuck-at-0.

Table 1.2. Tests for 3-input NAND gate.

A	B	C	D	Fault Class
1	1	1	0	A/0,B/0,C/0,D/1
0	1	1	1	A/1,D/0
1	0	1	1	B/1,D/0
1	1	0	1	C/1,D/0

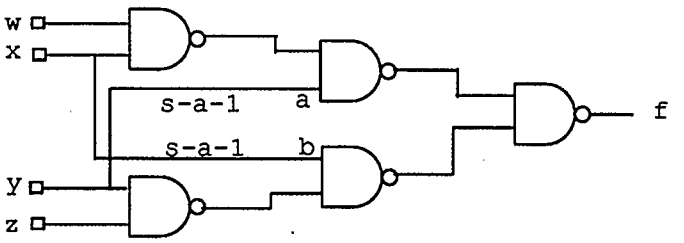

Figure 1.2. Two faults which are functionally equivalent.

Using the relations of equivalence and dominance allows many faults to be combined into a single class, reducing the number of faults to be considered in a complex system. A three-input NAND gate, therefore, will have four different fault classes and the tests for these faults are shown in Table 1.2. In the table, the fault consisting of one line l stuck-at-0 is shown as $l/0$.

The notion of equivalence and dominance can be applied to more complex circuits. Thus two faults which are in different parts of a larger circuit could possibly be equivalent. Figure 1.2 shows a simple circuit with four inputs and one output. Stuck-at-1 faults on the two lines marked a and b are equivalent, that is, the function under either faults is the same. However, equivalences such as these are more difficult to detect and, in practice, only equivalences and dominances around a gate are normally considered. More information on the concepts of the fault equivalence and dominance, as well as the idea of reducing the number of fault classes by fault collapsing, are found in [12, 18].

1.5 Summary

In this chapter, motivations that initiated the interest in the testing problem have been introduced. The cost of manufacturing a VLSI chip is shown to be very much affected by the testability figure of the chip. Design for testability, testability analysis programs, and new test generation algorithms are a normal consequence for the test process requirements.

The large number and complex nature of physical failures dictates that a practical approach to testing should avoid working directly with the physical failures. In most cases, in fact, one is not usually concerned with discovering the exact physical failure; what is desired is merely to determine the existence of (or absence of) any physical failure. One approach for solving this problem is to describe the effects of physical failures at some higher levels of abstraction. This description is called a fault model. The stuck-at fault model is the most popular for today's VLSI technology. The algorithms described in this work support this fault model.

CHAPTER 2

TESTING PROBLEM AND AUTOMATIC TEST GENERATION APPROACHES

In this chapter, the test generation problem is presented. Section 1 presents the test generation terminologies used throughout this work. The test problem complexity is identified and formulated in Section 2. The basic knowledge of testing concepts is presented in Section 3.

Although numerous approaches to test generation have been reported, only a few of these approaches are used in test systems. Section 4 presents some of these approaches, i.e., D-Algorithm, PODEM (Path Oriented DEcision Making), and FAN. These approaches are used as a reference for comparing our results with other work.

2.1 Definitions

Common terminology pertaining to test generation for logic circuits is readily introduced with an example. Figure 2.1 shows a combinational logic circuit and a test for a single stuck fault that causes node h to permanently assume a 0 state. A *stuck-at-1* (s-a-1) fault on a signal node causes that node to permanently assume the 1 state. A *stuck-at-0* (s-a-0) fault causes a permanent 0 on the faulted node. The five valued logic (0, 1, X, D, $\overline{D}$) is used to describe the behavior of a circuit with failures. The logic value D designates a logic value 1 for a node in the error free circuit and a 0 for the same node in the failing circuit, $\overline{D}$ is the compliment of D, and X designates a DON'T CARE value. A behavior difference between the good circuit

and the failing circuit propagates along a *sensitized path.* In Figure 2.1, the signal path h, j (the bolded line) is referred to as a sensitized path. Externally controllable nodes are referred to as *primary inputs.* Externally observable nodes are referred to as *primary outputs.* In Figure 2.1 assignment of the values 1, 1, X, X, 0 to the primary inputs a, b, c, d, e, respectively, constitutes a test for the fault h s-a-0.

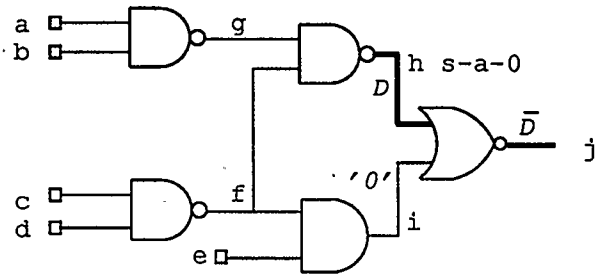

Figure 2.1. Example to illustrate test generation terminology.

Definition 1 : Two faults are said to be compatible if there exists at least one test vector which detects both faults.

Definition 2 : Two faults are said to be collapsed if the detection of one fault implies the detection of the other fault. The two faults can also be referred to as *indistinguishable* faults.

Definition 3 : The D-drive refers to the node with a logic value D or $\overline{D}$ and is used by the test generation algorithm to bring it closer to the primary outputs. In Figure 2.1, node h represents a D-drive to the test generation process. If at any time in the test generation procedure, more than one node carries the logic values D or $\overline{D}$, then we refer to these nodes as the D-frontier. The test generation algorithm picks up one of these nodes to drive the test process, i.e., selecting the D-drive node.

Definition 4 : The implication procedure refers to the process of using the implication rules of logic gates to propagate signal values at gate input nodes to their

output nodes. This procedure is used to check the implication of logic assignments made during the test generation procedure. The result is used as a guide to the next step in the test procedure.

Definition 5 : Consistency check is a procedure used by test generation algorithms to check if the previously made decisions meet some objectives set by the algorithm. The decisions made by the test generation algorithm are referred to as *inconsistent* if they don't meet the objectives set by the algorithm. It must be noted that these objectives vary during the test procedure.

2.2 Test Generation Problem

With the progress of VLSI technology, the problem of fault detection for logic circuits is becoming more and more difficult. In developing tests for digital circuits, the faults that will actually occur are unknown. Instead, test sets are developed to detect a specific set of faults.

2.2.1 Problem Formulation

As Goel [8] stated in his paper, the test generation problem can be formulated as a search of the n-dimensional 0-1 state space of primary input patterns of an n-input combinational logic circuit. For example, in Figure 2.2, g is an internal node and the objective is to generate a test for the stuck fault g s-a-0. The logic value at g can be expressed as a Boolean function of the primary inputs X_1, X_2, ..., X_n. Similarly, each primary output (y_j, $j = 1, 2, ..., m$) can be expressed as a Boolean function of the state on node g as well as the primary inputs X_1, X_2, ..., X_n.

Let

$g = G(X_1, X_2, ..., X_n)$

and

$$y_j = Y_j(g, X_1, X_2, \ldots X_n)$$

where $1 \leq j \leq m$ and $X_i = 0$ or 1 for $1 \leq i \leq n$.

The problem of test generation for g s-a-0 can be stated as one of solving the following set of Boolean equations:

$$G(X_1, X_2, \ldots, X_n) = 1$$

$$Y_j(1, X_1, X_2, \ldots, X_n) \oplus Y_j(0, X_1, X_2, \ldots, X_n) = 1$$

for at least one j, $i \leq j \leq m$ and $X_i = 0$ or 1 for $1 \leq i \leq n$.

The first equation implies that a s-a-0 fault is first excited to logic 1 (opposite to the stuck-at level), while the second equation implies that the change of the logic value at the fault location can be observed at the primary outputs. The set of equations for g s-a-1 are the same as above except that G is set equal to 0.

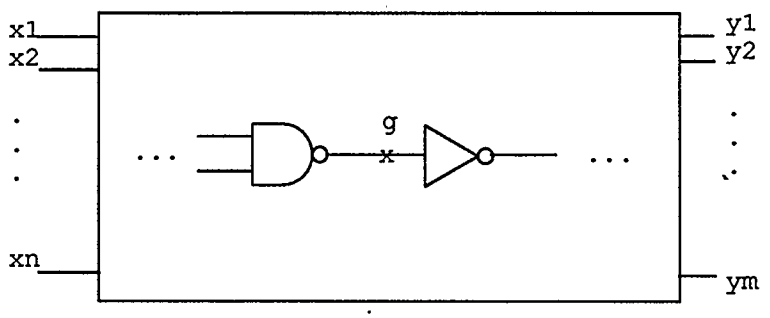

Figure 2.2. A Combinational circuit used in formulating test generation as an n-dimensional 0-1 state space search problem.

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 set of equations. More generally, the search will result in finding a k-dimensional subspace ($k \leq n$) such that all points in the subspace will satisfy the above set of equations.

2.2.2 NP-Completeness of Test Generation

The concept of NP-Completeness is used to prove that the amount of time required to solve a specific problem is beyond a certain practical limit [3]. The problem of test generation, which is known to belong to the class of NP-complete problems, can be viewed as a finite space search problem [8]. For a circuit with N primary inputs, there exists 2^N combinations of input assignments. Automatic Test Generation (ATG) algorithms basically search for a point in the primary input space that corresponds to a test pattern and consequently, to a solution of the search problem.

The NP-completeness property of the test generation problem necessitates that various heuristics be developed to create practical solutions for it. The PODEM [8] and FAN [5] algorithms are elegant examples in this regard. Many other fault analysis problems, such as the determination of the size of minimal test sets, coverage of multiple faults by single-fault test sets, and coverage of faults by randomly generated test sets are similarly besieged by their inherent complexity, and their solutions require thoughtful insights.

2.3 Elementary Testing Concepts

The three main concepts used by all test generation systems, namely, sensitization, consistency, and redundancy are described in this section. Although most of the following discussions are limited to combinational circuits and stuck-at-faults, these basic concepts can be extended for any digital circuit and any fault model.

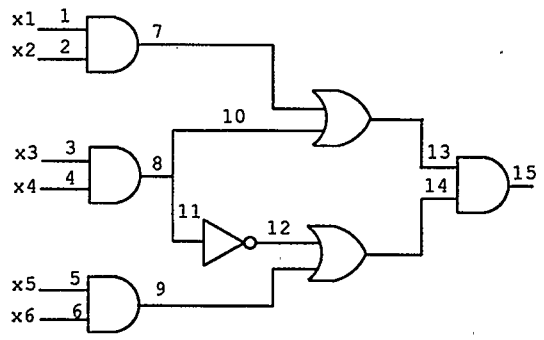

Figure 2.3. A sample circuit to describe sensitization.

2.3.1 Sensitization

Sensitization is a technique where a path consisting of many nodes is created to help propagate a stuck-at fault in a circuit. Searching the input space for a test pattern is equivalent to searching for a single (or multiple) sensitizing path.

Consider the circuit of Figure 2.3 and the fault 7 s-a-0. In order to detect this fault by a procedure that allows access only to the primary input lines (1, 2, 3, 4, 5, and 6) and the primary output line (15), it is essential that a test vector must somehow create a change on line 7 and ensure that the change can be seen on line 15. That is, the test vector must produce a 1 on line 7, and line 15 should be *sensitized* to line 7 in the sense that the output created on line 15 clearly shows whether the signal on line 7 is 0 or 1. If the path from line 7 to line 15 is traced in Figure 2.3, the first condition for sensitization is that line 10 be a 0. Indeed, if line 10 is a 1, then line 13 would be 1 irrespective of the value on line 7. In other words, a 1 on line 10 would desensitize line 7 to line 13. Moreover, since there is no other path to transmit the value on line 7 to line 15, line 10 being a 1 will also desensitize line 7 to line 15. Thus

assuming that line 10 is a 0, the next condition for the sensitization is that line 14 be a 1. If both of these conditions exist in the circuit, then when a 0(1) is applied to line 7, the circuit output is going to be a 0(1). In other words, any input vector that can create a 1 on line 7, a 0 on line 10, and a 1 on line 14 will in the fault free circuit produce a 1 on the output line, and in the faulty circuit a 0 on the output line, and will, therefore, be a test vector for 7 s-a-0.

The concept of sensitization needs to be explained further in the situations involving more than one path from the faulty line to a primary output line, and in the case of multiple stuck-at faults. In summary, the concept of sensitization is fundamental to understanding how a fault is detected from the input and output lines only. However, the process of determining a sensitized path(s) in a general situation is not a simple procedure.

2.3.2 Consistency

As shown above, some logic assignments and conditions are needed to carry out the sensitization process. However, just formulating such conditions does not always guarantee that an input vector satisfying such conditions also exists. Thus, formulating conditions to create a change and to propagate the change along a sensitized path is just one step. The second equally important step is to determine which, if any, vector(s) satisfies such conditions. When this process is carried out by exploring the circuit structure, it is often referred to as the line justification or consistency process.

In general, consistency, or line justification is not a simple process since different conditions may result in contradictory requirements. For example, consider the fault set (7 s-a-0, 12 s-a-1) of Figure 2.3. The fault is tested by creating changes on both the faulty lines, and the change is propagated only from line 7. This will require

that the following conditions be held: line 7 is 1, line 12 is 0, line 10 is 0, and line 9 is 1. However, since line 10 and 12 require the same binary signal, but can have only complimentary values, it is clear that no vector will satisfy this particular set of conditions.

An ideal line justification algorithm will, at each step, make a decision that will not have to be changed. In general, however, this is not possible since making an irreversible decision requires knowledge which is not available at the time of decision and can be obtained only by reversing the decision and starting again. The most one can do in this situation is to use some insights or heuristics so that as few decisions as possible are changed. Actually, due to this decision process, the test generation problem is NP-complete [5].

2.3.3 Redundancy and undetectability

A fault is said to be undetectable if there is no vector to detect this fault, and the line associated with the fault is called a redundant line. For instance, in the trivial circuit of Figure 2.4, the fault 5 s-a-1 is undetectable, since sensitizing it would require that each of lines 3, 4, and 6 be a 1, implying in turn that $x_1 = 1$, $x_2 = 1$, and $\overline{x_1.x_2} = 1$. These being contradictory requirements, one can conclude that if 5 s-a-1 existed in the circuit, then as far as the input/output behavior is concerned, the circuit is going to behave as if there is no fault in it. Such undetectable fault seems to be harmless when not probed further. However, as previous research in the area has shown, one must know where the redundant lines in the circuit are, to be able to carry out an effective detection of the detectable faults. For example, in the circuit of Figure 2.4, the input vector (1, 1, 0) is a test vector for a 1 s-a-0. However, in the presence of of the undetectable fault 5 s-a-1, (1, 1, 0) cannot test 1 s-a-0. Thus,

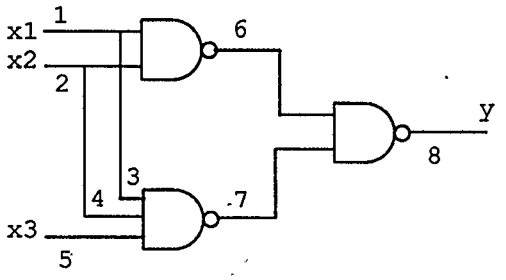

Figure 2.4. Example of redundancy.

an undetectable fault can invalidate the testing of some detectable faults if both are present simultaneously.

Another effect of an undetectable fault is its impact on the test generating efforts for a given circuit and a fault set. If a fault set is undetectable, any resources spent in trying to obtain a test vector are wasted. It is thus useful to remove all the undetectable faults from the fault set before the test generation step. As it turns out, even the process of determining whether a fault is detectable or not is as complex as the test generation process which is NP-complete. The best hope, therefore, is to avoid the appearance of redundant lines during the design phase of the circuit under consideration.

2.4 Test Generation Approaches

2.4.1 Random Test Generators

The concept of generating test vectors for a digital circuit by some random process probably provides the simplest approach to the test generation problem [1, 20]. The major current issues for random test pattern generation are: selecting the test length, determining the fault coverage, and identifying random-pattern resistant faults (faults that are hard to detect with random patterns). These could, in principle, all be accomplished by a full single-stuck fault simulation of the network to be tested [21]. The

development of special-purpose equipment is decreasing the cost of fault simulation. Despite this cost reduction, full fault simulation remains expensive for large circuits that require long random test sequences for adequate fault coverage.

The only viable alternative to full fault simulation appears to be the use of a probabilistic model of random test generation [13]. Probabilistic methods do not give exact fault coverage values, but they do provide more insight into the relations between circuit characteristics and test parameters.

2.4.2 Deterministic Test Pattern Generators

The problem of deterministically generating a test pattern for a given fault is to find a combination of assignments of logic values (0 or 1) to the primary inputs which:

- excite the target fault,
- monitor the target fault at, at least one of the primary outputs.

Since the properties of deterministic test generation fulfill the requirements for a systematic search problem, automatic test generation algorithms usually build a decision tree and apply a backtracking search procedure [5, 8], in order to find a solution for problem.

The D-algorithm [16, 17] is probably the most known test generation algorithm. It develops 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. In this calculus, each line can be either a 0, 1, X (unknown), D, or $\overline{D}$. The faulty line is assigned a D or $\overline{D}$ depending on the fault on the line. The next step is to use the calculus and the circuit structure information to determine values on the other lines so that the

D or $\overline{D}$ can be sensitized to the primary output line. A line justification step is then carried out to justify the values assigned in the preceding step. Both the sensitization and line justification steps may have to be carried out many times before a test vector is obtained.

The PODEM (Path-Oriented Decision Making) algorithm was introduced in particular to perform better than the D-algorithm for circuits containing mostly Exclusive-OR gates. It was, however, demonstrated to have a better performance than the D-algorithm for various other types of circuits as well. The approach taken by PODEM appears to be the first to treat the test generation problem as a classic branch-and-bound problem. More fundamentally, 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 line. 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.*

The branching procedure basically consists of making an intelligent choice on the primary input line to be selected and the binary value to be assigned to it. If, however, at any time in this branching, an inconsistency is determined, the branching stops and bounding starts. The Pi line which was most recently assigned a binary value is assigned the complimentary value, and branching is started again. If, however, both the values on the most recent line result in a bounding step, then the Pi line next to the most recent is treated in a similar manner. The complete process stops when either a test vector is found or when the fault is determined to be undetectable. A high level description of the PODEM algorithm is shown in Figure 2.5.

The PODEM algorithm uses a two-step process to choose a primary input (Pi) and logic level for initial assignment.

```
Procedure podem( ) {
   assign(pi);
      /* assign a binary value to unassigned primary input */
   implication( );
      /* apply implication rules to all nodes in D-frontier */
   if a test generated; exit done;
      else if no more combinations of pi's exist; exit undetectable fault.
      else set untried combinations of values on unassigned pi's.
      start branch step.
}
```

Figure 2.5. High level description of PODEM.

Step 1: Determine an initial objective. The objective should be to bring the test generator closer to its goal of propagating a D or $\overline{D}$ to a primary output.

Step 2: Given the initial objective, choose a Pi and a logic level such that the chosen logic level assigned to the chosen Pi has a good likehood of helping towards meeting the intial objective.

The following two simple propositions are evaluated to carry out the process of determining if a test is possible with additional assigned Pi's.

Proposition 1: The signal node (for the given stuck fault) has the same logic level as the stuck level.

Proposition 2: There is no signal path from an internal signal node to a primary output such that the internal signal node is at a D or a $\overline{D}$ value and all the other nets on the signal path are at X.

If Proposition 1 is true, then it is obvious that assignment of known values to unassigned Pi's cannot result in a test. If Proposition 2 is true, then there exists no

signal path along which the D or the $\overline{D}$ can propagate to a circuit output. Hence, only if both Propositions 1 and 2 are false, then it is concluded that a test is possible with the present Pi assignment even though further enumeration may show that it is not.

Table 2.1 shows a comparison of PODEM and D-algorithm [8]. It shows that PODEM achieved 100 percent test coverage on each of the four Error Correction And Translation (ECAT) type circuits used. Since DALG (D-Algorithm) and PODEM are complete algorithms, given enough time, both will generate tests for each testable fault.

Table 2.1. Comparison of PODEM and DALG.

Circuit type	No. of gates	Normalized run time		Test coverage	
		PODEM	DALG	PODEM	DALG
ECAT	828	1	34.5	100.0	99.70
ECAT	935	1	12.8	100.0	93.10
ECAT	948	1	5.7	100.0	95.70
-	951	1	2.2	99.50	99.50
-	1249	1	3.5	98.20	98.20
-	1172	1	2.6	98.50	98.50
ALU	1095	1	15.3	96.50	96.20
MUX	1082	1	3.2	96.60	96.60
-	915	1	3.9	96.30	96.30
DEC	1018	1	3.8	99.10	99.10
PLA	538	1	2.5	94.50	94.50
PLA	682	2.6	1	89.40	89.40
PLA	1566	1	3.1	97.40	97.40

From the above discussions, it is obvious that to accelerate an algorithm for test generation, it is necessary to reduce the number of occurrences of backtracks (branching-bounding cycles) in the algorithm and to shorten the processing time between backtracks. Based on that, the FAN [5] algorithm started with the basic conjecture that the PODEM does not fully exploit the excellent framework in which

it works. FAN has employed a better heuristic in the bounding-and-branching steps to speedup the test generation process. Table 2.2 shows the results obtained on active sample circuits by implementing both FAN and PODEM. The results show that FAN is more efficient and faster than PODEM. The average number of backtracks in FAN is lower compared to that of PODEM.

Table 2.2. Comparison of FAN and PODEM.

Circuit	No. of gates	Normalized run time		Test coverage		Ave. No. of backtracks	
		PODEM	FAN	PODEM	FAN	PODEM	FAN
ECAT	718	1.3	1	99.20	99.52	4.9	1.2
ALU	1003	3.6	1	94.25	95.49	42.3	15.2
ALU	1456	5.6	1	92.53	96.00	61.9	0.6
ALU	2002	1.9	1	98.75	98.90	5.0	0.2
ALU	2982	4.8	1	94.38	96.81	53.0	23.2

2.5 Summary

In this chapter, the test generation problem has been presented and formulated. It has been shown that the test generation problem is a complex problem and is considered to be NP-complete. Different approaches have been used to tackle the test problem, either by randomly generating test vectors or by using other deterministic test generation methods. Test generation, as a space search problem, has evolved in designing efficient algorithms as the case in PODEM and FAN. Most of the test systems reported for this decade are based on these two algorithms. SOCRATES [19] algorithm, for instance, is an improved version of FAN. Based upon the sophisticated strategies of the FAN algorithm, an improved implication procedure, an improved unique sensitization procedure, and an improved multiple backtrack procedure are described. In general, however, most of the work in the testing area is useful under

very specific circumstances. Despite the steady growth in the area of digital system testing, it has yet to witness the development of a consistent framework which can provide efficient testing algorithms for large and complex systems.

CHAPTER 3

AN EFFICIENT ALGORITHM FOR TEST GENERATION

The problem of test generation is known to be NP-complete [6, 10]. The present approaches for test generation are time–inefficient for large size combinational circuits. In this chapter, we present an efficient algorithm for test generation. In our algorithm, the test generation problem has been formulated as a model of a global search of the sensitizing paths between the primary input and output nodes. Such a simple model offers significant improvements in efficiency for the test generation of large combinational circuits.

3.1 Introduction

A basic problem in testing is to determine an optimal testing procedure for a given circuit and determine a set of faults which models most of the failures likely to occur in that circuit. A typical testing procedure consists of three steps: test generation, test application, and test verification. The performance of a testing procedure is often measured in terms of the time and effort required to carry out these three steps. The time taken to generate a single test pattern grows exponentially with the number of inputs in the circuit. Most deterministic test pattern generation algorithms exhaustively search the circuit netlist for test vectors to detect stuck-at faults. To improve an algorithm's efficiency, some limitations are imposed on the procedure to minimize the number of backtracks and thus the test generation time. A test vector

is generated for a target fault and then a fault simulation is applied to check for the other faults which can be detected by the same vector. Consequently, the number of generated vectors are much less than the number of modeled faults.

A significant theoretical study [11] suggests that no test generation algorithm for combinational circuits with a polynomial time complexity is likely to exist. Goel [7] argues that the time for a complete test generation must grow at least as the square of the number of gates in the circuit. Presently most well-known algorithms make use of the path sensitization idea in one form or the other. Among all, the D-algorithm is the oldest and the best-known algorithm. The performance degradation of the D-algorithm arises due to an excessive amount of backtracking. Based on this observation, Goel used a branch and bound technique and devised a new test generation algorithm called PODEM [8], described in chapter 2. Fujiwara and Shimono [5] described a technique to further accelerate a path-sensitization algorithm like PODEM. Their algorithm, called FAN, makes an extensive analysis of the circuit connectivity in a preprocessing step to minimize the amount of backtracking. Schulz et al. [19] further improved the performance of FAN by improving the implication procedure and built a test generation system called SOCRATES. They also described a unique sensitization procedure and an improved multiple backtrace procedure.

A number of researchers have attempted to reduce the test generation time. However, no algorithm is efficient for the general class of faults. Each fault may need different heuristics for efficient test generation. Using different strategies for test generation lowers the testing time [14]. The concept of fault partitioning has been used in a parallel multiprocessor system to achieve fast test generation for large circuits. Patil and Banerjee [15] have shown that large speed up factors can be achieved by using a dynamic fault partitioning algorithm.

Most approaches discussed above suffer from exponential time complexity in the worst case. It is generally believed that, for combinational circuits, the time taken in the current test generation approaches is prohibitive.

In this chapter, we present an efficient test generation algorithm for combinational circuits. The test generation problem has been formulated as a model of a global search of sensitizing paths between primary input and output nodes. The algorithm generates test vectors with at least 98% fault coverage of all testable faults in a circuit in a short time. Hence, it can be used to enhance the overall test generation procedure by replacing the traditional random test generation technique which has a limited test coverage of only 60% to 85%. Moreover, our approach ensures a more compact test set than that of the random test generation technique.

The rest of this chapter is organized as follows. In section 2, the motivations that initiated the development of our algorithm is presented. New ideas for a target fault switching technique during the search procedure are also discussed. In section 3, we describe the test generation problem in the context of our approach. An efficient test generation algorithm is introduced in Section 4. Finally, section 5 summarizes this chapter.

3.2 Motivations

Motivation for this work comes from the observation that test generation for the detection of faults in digital circuitry can be made more efficient by employing a target fault switching technique in the search process [22]. For more elaboration, let us consider some of the strategies used in PODEM, and explained in chapter 2, in the process of searching for a test pattern for a specific target fault. Although the following discussion is concerned with PODEM, it also applies to other algorithms

such as FAN and SOCRATES.

In PODEM, backtracking is the most computationally expensive step in the process of searching for a test vector. Backtracking refers to a branch procedure terminated by a bound step. 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. Backtracking stops when either a test vector is found or when the fault is determined to be undetectable. A fault is declared undetectable if the number of occurrences of backtracks exceeds a limit specified by the user. Therefore, a fault classified as undetectable in most deterministic ATPG algorithms is either a testable fault which needs more backtrack steps to be detected or a truly redundant fault.

One can conclude that if f_1 is a target fault in PODEM, then a number of backtrack steps occur before the fault is detected or declared undetectable. Suppose that the average number of backtracks made by PODEM to detect a fault set in a circuit is N. Then on the average, only one backtrack step is used to generate a test vector for the fault f_1, while N-1 steps are used to assign logic values that lead to a bound step. As a result, some logic assignments associated with the search tree before each bound step are rejected, which is a waste of computer time. A better approach is to use these logic assignments to direct the test procedure to detecting another target fault. Therefore, the logic assignments that do not help in detecting the fault f_1 could be used to detect another fault f_2, where f_1 and f_2 are distinguishable faults. To achieve that, the test generation algorithm needs a procedure to switch the search process to different target faults whenever a bound step occurs. Since the problem of searching for distinguishable sets of faults is as difficult as the test generation problem, we may only consider a new target fault which meets the objective of propagating a D or

$\overline{D}$ closer to a primary output using the logic assignments made before the bound step. Heuristics can easily be employed to achieve this purpose, for instance, a new target fault search procedure might use the circuit topology to search for those faults which are topologically nearby the original target fault. Another approach might use a random selection of the new target fault from the fault list.

It is not necessary to implement such procedure in the context of the existing ATPG algorithms like PODEM or FAN. In this work, we are taking a global testing approach to implement a non-target fault algorithm. In our approach, a tree of fault assignments is created in a similar manner to the depth - first search algorithms starting from primary outputs.

3.3 Test Generation Viewed As A Global Search Problem

Most deterministic Automatic Test Pattern Generation (ATPG) algorithms view the test generation problem as a search of the n-dimensional 0-1 state space of primary input patterns of an n-input combinational logic circuit. Because of the complexity of the test generation problem (i.e. NP-complete), search heuristics are used to limit the exponential time complexity of these algorithms. For example, PODEM is an implicit enumeration algorithm in which all possible primary input patterns are implicitly examined as tests for a given fault. Implicit enumeration refers to a subset of the branch and bound algorithms designed specifically for search of an n-dimensional 0-1 state space. Although new heuristics are developed to improve the algorithm's performance such as those employed in FAN and SOCRATES, significant improvement in timing complexity is not yet achieved. From our study of these algorithms we believe that the search for a test vector to detect a single target fault yields time – inefficient algorithms. We suggest that a switching technique to different target faults during the search process is necessary.

In this work, the problem of generating test vectors without targeting a specific stuck-at fault set is addressed. An attempt is made to solve this problem by applying certain logic assignment rules. These rules consider the representation of a single stuck-at fault as a number of fault patterns at the primary inputs. A fault pattern refers to a combination of fault assignments (D and $\overline{D}$) at the primary inputs. For example, a two-input NAND gate with fault D at its primary output can be represented as one of three patterns ($\overline{D}$, $\overline{D}$), (1, $\overline{D}$), ($\overline{D}$, 1) at the primary inputs.

In our approach, a depth - first back fault assignment is conducted starting from the primary outputs, where each output is selected in some arbitrary order. The depth - first back assignment procedure is terminated at the primary inputs, we refer to this process as output cones sensitization. The test generation problem can then be modeled as one involving a global sensitization of paths between the primary input and output nodes. The object of this work is to show a cost - effective method to implement this model.

3.4 An Efficient Algorithm for Test Generation

The problem of searching a combinational logic circuit for fault patterns that represent a fault on an internal node consists of two parts: defining the back fault assignment rules (referred to as B-rules) and implementing a back assignment procedure similar to the depth - first search algorithms.

3.4.1 Back Fault Assignment Rules (B-rules)

The object of the B-rules is to allow the fault assignment of logic values D and $\overline{D}$ to appear on each node in a circuit at least once during the back assignment procedure. Consequently, the resultant fault patterns at the primary inputs will include at least a single pattern which sets an internal node to the logic value D or $\overline{D}$. In other

words, the B-rules guarantee that any stuck-at fault can be excited by some of the generated fault patterns. Figure 3.1 defines the B-rules used in our algorithm. A fault is supposed to exist on a gate's output node, and the fault representation is carried out at the gate's input nodes.

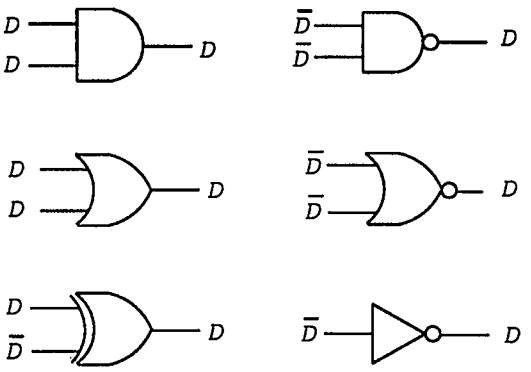

Figure 3.1. The back fault assignment rules.

For example, a fault D on the output of a NAND gate is expressed as the ($\overline{D}$, $\overline{D}$) fault pattern at its input nodes. Similarly, a fault $\overline{D}$ on a NAND gate output node is expressed as the (D, D) fault pattern at its input nodes. There are other representations that can be used, as shown in section 3, but the B-rules ensure that only logic assignments D and $\overline{D}$ are used in our approach. Accordingly, the generated fault patterns will represent a D-frontier to all circuit nodes.

3.4.2 The Back Fault Assignment Procedure

Consider the problem of generating the fault patterns for a fault D on an internal node g as shown in Figure 3.2. There are L paths by which node g can be reached from the primary inputs. The region which comprises the sensitizing paths between

node g and the primary inputs will be referred to as the output cone of node g. Each sensitizing path in the output cone of node g has a length which represents the number of logic gates along the path from node g to the primary inputs. For example, path1 has a length equals one, while path2 has a length equals two, and so on. The back fault assignment procedure arranges the circuit nodes in different levels in the following way. If the output node of a gate belongs to an assignment of level L, then all the input nodes of this gate belong to level $L+1$. This situation is similar to the two-level circuit configuration with the exception that gates on the same level may not be of the same type.

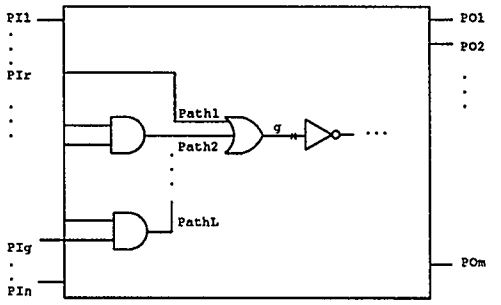

Figure 3.2. A circuit example to describe testing as a Global search problem.

Now consider the case where node g has a fault logic value equals D. Then using the B-rules described earlier and starting from node g, the back fault assignment procedure sweeps the output cone of node g by propagating the fault D toward the primary inputs. The back fault assignment tree grows in a similar manner to the depth - first search algorithms. As the back assignment tree builds up, it creates fault assignments at each node in the output cone of node g. The back propagation

of faults is terminated at the primary inputs. At this point, the output cone of node g is said to be *regionally sensitized.*

Since different sensitizing paths are not equal in length, some primary inputs will be assigned earlier than others. Therefore, the back fault assignment procedure checks at each level of assignments for the primary inputs that have been assigned new fault values. The procedure codes the primary inputs as a new fault pattern. Consequently, each new collected fault pattern corresponds to an added sensitizing path with all its nodes assigned to either D or to $\overline{D}$. Hence, a rough estimate of the possible number of generated fault patterns would be the difference between the maximum and the minimum lengths of two sensitizing paths.

Now consider the case where g is a primary output node. Then the process of generating the fault patterns by regionally sensitizing the primary output cones is referred to as *global sensitization.* A high level description of the global sensitization algorithm, *glob_sens*, is shown in Figure 3.3. To illustrate the idea of the *glob_sens* algorithm, we use a sample circuit C17 selected from the ISCAS'85 benchmarks and shown in Figure 3.4.

The algorithm *glob_sens* first selects a primary output node and assigns it a D logic value. Assume that *glob_sens* arbitrarily selects the primary output node 10 in Figure 3.4. Node 10 now represents the only currently assigned node in a node list of assignments. Then according to the B-rules, both inputs of gate G_5 are assigned a similar logic value $\overline{D}$. At this point, a new level of assignment is created which includes a list of two nodes 6 and 9. The algorithm *glob_sens* keeps track of the node numbers in the new list of assignments, the length of this list, and the associated logic values, as depicted in Figure 3.3. Starting at node 6 then node 9, the *glob_sens* algorithm assigns the logic value D to the nodes 1, 2, 3, and 7. The first three

```
Input   : A circuit's netlist.
Output : A set of test vectors.

Procedure glob_sens( ) {
   select a primary output node and assign a D logic value to it;
   for each element in the node list {
      /* list of previously assigned nodes */
      execute the back-assignment function;
      return the new list of node assignments;
      return the length of the new list;

   }
   if at least one PI appeared in the list {
      collect a new test vector;
   if the length of the new list is zero {
      /* only PI nodes appear in the list*/
      get the next PO;
   if the PO list is empty {
      exit
         }
      }
   }
}
```

Figure 3.3. The Algorithm to globally sensitizes output cones.

nodes belong to the primary input, hence, an input fault pattern (D, D, D, X, X) is generated. The three nodes 1, 2, and 3 are then removed from the node list leaving node 7 to be the only element in the node list at this level of assignment. Accordingly, the fault D at node 7 is propagated using the B-rules to assign the primary inputs 3 and 4 a logic value of $\overline{D}$. The last step produces a change at only two primary input lines (3 and 4), and hence provides another fault pattern $(D, D, \overline{D}, \overline{D}, X)$. The *glob_sens* algorithm repeats this procedure for each output cone in a circuit.

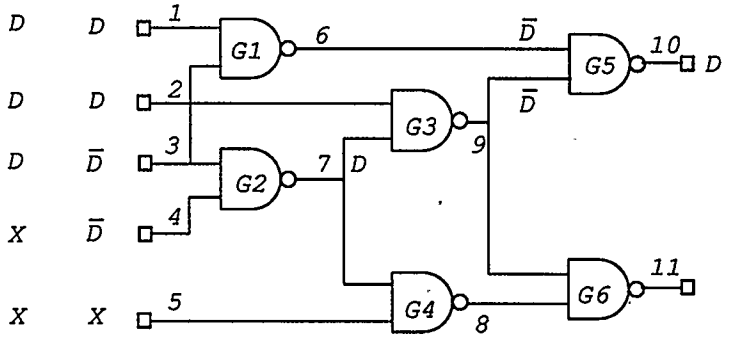

Figure 3.4. Example circuit C17 shows how the back propagation procedure assigns fault values to the circuit nodes.

The *glob_sens* algorithm either substitutes the X terms with a logic value D, $\overline{D}$, or simply keeps X unchanged. If the node with the *don't care* term shares the output cone under which the back propagation process is occurred as an input to a gate, the algorithm checks for the other gate's input logic value and assigns it to that node. For example, node 4 in the first generated fault pattern has an X logic value. Node 4 is also an input to gate G_2, which belongs to the first output cone. The logic value assigned to node 4 will be the same as that of node 3 which is D. On the other hand, the logic value of node 5 is left unchanged because it is an input to gate G_5 which does not belong to the first output cone. In this way, we can avoid generating a large number of redundant fault patterns since each output cone is dealt with separately.

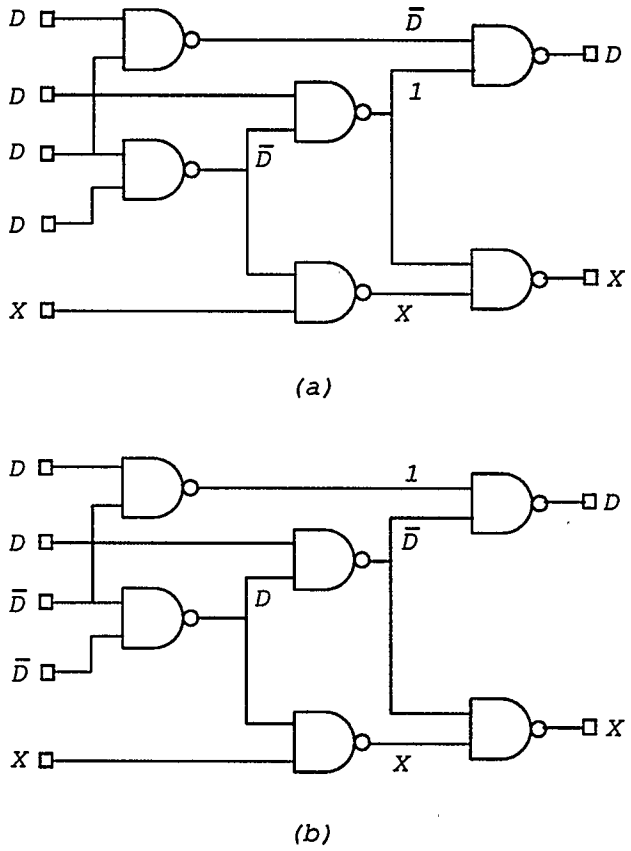

Figure 3.5. The implication procedure of the resultant fault patterns (a) (D, D, D, D, X) (b) $(D, D, \overline{D}, \overline{D}, X)$

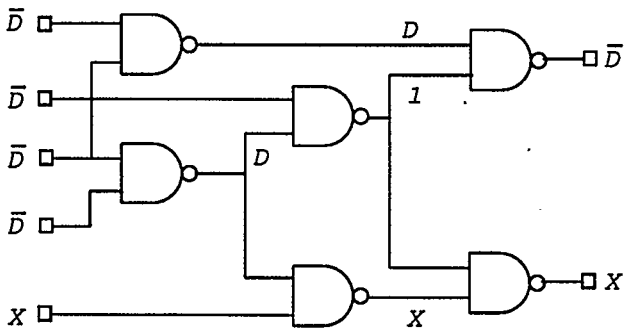

Figure 3.6. The implication of the fault pattern $(\overline{D}, \overline{D}, \overline{D}, \overline{D}, X)$.

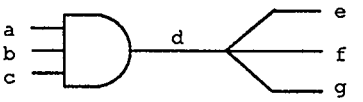

Figure 3.7. Fan-out point example.

Now let us consider the implication of the resultant fault patterns on the circuit under consideration. Figures 3.5.a and 3.5.b show the implication of the two fault patterns (D, D, D, D, X) and $(D, D, \overline{D}, \overline{D}, X)$, respectively. The first pattern sensitizes a path from the primary inputs to the primary outputs through node 6, while the other pattern sensitizes the path through node 9. The two patterns combined together sensitize the output cone of the primary output node 10. As mentioned earlier, the *glob_sens* algorithm guarantees that each node must be assigned a complimentary logic value at least once in the back propagation process. This is simply done by taking the compliment of the resultant fault patterns. Figure 3.6 shows the circuit C17 with an input fault pattern $(\overline{D}, \overline{D}, \overline{D}, \overline{D}, X)$, which is the compliment of the first generated fault pattern. By implication of this pattern, the same sensitizing paths exist but with each internal fault logic value complimented.

An important point is the behavior of the *glob_sens* algorithm with fan-out points as the one shown in Figure 3.7. In case of nonconvergent fan-out, faults at the fan-

out points may be collapsed because any test that detects a fault along one of the destination lines from the fan-out point will also detect the corresponding fault on the origin line of the fan-out point [18]. For example, in the circuit shown in Figure 3.7, any one of the faults e s-a-1 (s-a-0), f s-a-1 (s-a-0), or g s-a-1 (s-a-0) is collapsed with d s-a-1 (s-a-0). That is, the vectors which detect these faults will also detect the corresponding fault on line d. In this case, in the back fault assignment procedure, if all the nodes e, f, and g are assigned the same logic value at the same level of assignment, then the algorithm discards two of these values and holds only one logic value and assigns it to node d. In case of conflict, i.e., two nodes being assigned different logic values at the same level of assignment, the algorithm holds both values and assigns them to node d. Consequently, node d is replicated in the node list with two different logic values. As a result, when the logic assignments from node d land on the primary input nodes, two fault patterns are generated with only the primary inputs involved in the output cone at node d, assigned complimentary logic values. For convergent fan-out points, the algorithm behaves in a similar manner.

3.4.3 Algorithm Constraint

Due to the connectivity of the logic circuits, some nodes may be assigned a logic value more than once. At a fan-out node, for example, different paths are connected together and the node may have multiple assignments in the node list. Therefore, the algorithm keeps track of the number of assignments for each node and the associated logic values. In a circuit, the number of assignments per level may increase dramatically. In the worst case, the computing time for generating the fault patterns may grow exponentially. To limit the exponential growth of the size of the assignment tree, a parameter K is used in the algorithm to limit the maximum number of assignments

for a node in the circuit. The parameter K serves as a tap which limits the number of fault patterns released by the circuit and thus controls the time taken to generate these vectors.

The advantage of limiting the maximum number of assignments per node is twofold. First, this will dramatically decrease the test generation time. Second, compared to the total number of circuit nodes, the size of the arrays storing the dynamic node lists and the corresponding logic values can be kept small. This is due to the fact that only one assignment level is considered at a time. Hence, the size of an array is much smaller than that of a circuit. In fact, for a wide range of K, the size of the arrays never exceeds the number of nodes in the tested circuits. On the other hand, low values of K degrades the fault coverage of the test set.

3.5 Summary

The current test generation algorithms can generate test vectors for complex combinational circuits and guarantee 100 percent coverage for testable faults. However, the test generation time increases exponentially with the circuit complexity. A new approach which combines simplicity and fast performance has been developed for the test generation of large combinational circuits. The similarities of our algorithm with current approaches has been identified. The test generation problem has been formulated as a problem of a global search of sensitizing paths between the input and the output primary nodes. To achieve this objective, a circuit environment is created such that all faults which can be detected by multiple - path sensitization are sensitized. The algorithm may not detect all faults that are sensitized with only a single path. Hence, the fault coverage limitation will only be a function of the circuit complexity.

CHAPTER 4

A MODEL FOR PARALLEL TEST SYSTEM

As parallel processing hardware becomes more common and affordable, multiprocessors are being increasingly used to accelerate VLSI CAD algorithms. The problem of partitioning faults in a parallel test generation/ fault simulation environment has received very little attention in the past. In a parallel test system environment, the fault partitioning method used can have a significant effect on the overall test length and speedup. Also, for efficient utilization of available processors, the work load has to be balanced at all times. In this chapter, the model of a parallel test system is presented. The test generation algorithm described in chapter 3 is employed in the parallel test system. A partitioning method for load balance over processors is also described.

4.1 Introduction

Parallel test generation is a technique where multiprocessing hardware is used to achieve order of magnitudes speedup for test generation. A fault/circuit partitioning step precedes the test generation process. Static and/or dynamic fault/circuit partitioning can be used in assigning loads to different processors. In static partitioning, an equal number of faults or partitions of a circuit are assigned to each processor. It has been shown that each set of faults which belongs to the same processor should be a compatible fault set [15]. By definition, every two compatible faults in a fault set

will have at least one test vector that detects both faults. Consequently, assigning compatible set of faults for each processor yields a time - efficient algorithm. Unfortunately, the problem of generating a compatible set of faults is as hard as the test generation process itself.

Most parallel test generation systems use a test generation/fault simulation environment. Each generated vector from one processor is simulated by all other processors to minimize the fault list at each processor. Therefore, an equal number of faults loaded to each processor is a poor estimate for load distribution. On the other hand, dynamic fault partitioning alleviates this problem by keeping track of the idle processors and the busy ones. It dynamically redistributes part of the undetected faults to the idle processors. This procedure involves massive communications between the processors for overall efficient performance. Although a much better load balance over processors can be achieved by dynamic partitioning, extensive process communication among processors may contribute significantly to the overall run time.

Another major issue in parallel test generation systems is the test length. The test length depends on both the deterministic pattern generator and the partitioning method. Random test pattern generators produce a large number of test vectors, while the fault coverage is limited to 60%–85%. On the other hand, deterministic test pattern generators are producing a small number of vectors with a very high fault coverage at the expense of high time complexity. Therefore, deterministic test generators have been incorporated in parallel test systems.

4.2 Parallel Test System Environment

The design of a parallel test system involves a trade-off between the test length and the cut off communication factor for the processors. If two processors are not

communicating, they might end up doing the same job, adding time and space overheads. For example, let (f1, f2) and (f3, f4) be two sets of faults with the first set assigned to processor P1 and the second set assigned to processor P2. Then let f1 and f4 be compatible faults, i.e., there exists at least one vector which detects both faults. Processor P1 will generate a test for fault f1 while P2 does the same for fault f3. If the two processors are not communicating by sending the generated vectors to each other for fault simulation, processor P2 will generate another vector for fault f4. Hence, a test length of four vectors will be generated, while only three were sufficient.

An adequate solution to the above problem is the use of a parallel test system with the test generation phase made separately without the fault simulation step. This will only be feasible if the test pattern generator incorporated in the test system generates an optimal or a complete set of vectors. As has been indicated in chapter 3, the proposed algorithm generates a set of test vectors without concurrent fault simulation. That is, the test set is first generated and then fault simulation is applied. The test set might include redundant vectors, but so far the test length is comparable to those generated by other deterministic pattern generators, such redundancy could be tolerated. We shall show through experiments that the test length of our algorithm is in general less than those generated from other deterministic algorithms. In order to incorporate our algorithm in a parallel test system, the fault/circuit partitioning step must be carried out first.

4.3 Partitioning Algorithm

A clear advantage of the proposed algorithm is that only static partitioning of output cones can be used. It is an advantage because it cuts off process communication and saves the overhead time of signals between processors. For the static partitioning method, an accurate estimate of time units needed to balance the loads over the

available number of processors is crucial. We have chosen the number of node back-assignments made by the test generation algorithm as an accurate estimate for the test generation time. Since we are optimizing the system for an overall minimal run time, each processor has been assigned a load of equal number of back- assignments. The partitioning procedure is shown in Figure 4.1.

The partition algorithm first calculates the number of assignments associated with each output cone. As shown in the partition procedure, to ensure an accurate calculation of the number of node assignments, a similar procedure to the one used in the test generation process has been implemented. The partition procedure groups the output cones and divides them among processors according to the number of node assignments associated with each cone. The above procedure minimizes the difference in the number of node assignments between the largest and the smallest element in the resultant group, as depicted in the *for* loop of the procedure. The fact that the procedure starts with the group containing the output cones with maximum number of assignments helps in minimizing the differences in the elements of the resultant group as the algorithm steps forward.

For example, consider a circuit with 10 primary outputs and the available number of processors N_p equals 4. If the number of node assignments in the 10 output cones are 1, 2, 2, 3, 5, 6, 10, 16, 17, and 20, with N_p equal to 4, two groups of 4 elements and a residual group of 2 elements are created. The first group which includes the output cones with the largest numbers of assignments will hold the elements 10, 16, 17, and 20, while the second group will hold the elements 2, 3, 5, and 6. Executing the *for* loop in the partition procedure results in a new group which holds the elements 16, 21, 20, and 22, which is going to be 16, 20, 21, and 22 after sorting the group. The final step is the addition of the residual group to the last two elements in the

```
Input   : Circuit netlist.
        : Number of gates, PI nodes, and PO nodes.
        : Number of available processors (Np).
Output : An ordered assignment sequence.

Procedure partition( ) {
  calculate the number of expected back-assignment for each output cone.
  if the PO list is empty; sort the numbers list;
    else get next PO.
  divide the list into groups with each group having Np elements;
  if Np/Npo is not integer;
    create a residual group with the remaining number of elements.
    /* Npo is the number of primary output nodes. */
    else return the number of groups;
  let group number i be the one with the maximum number of assignments;
  for each element in group i {
    add element j of group i to element Np-j of group i+1;
    save result in group i+1;
    sort group i+1;
  }
  repeat the last step until groups are exhausted, including the residual group (if any)
  assign elements in the final group to Np processor.
}
```

Figure 4.1. The partitioning algorithm.

forehead group. The final group will hold the elements 18, 21, 21, and 22. These elements represent the load distribution over a set of 4 processors. The difference between the largest and the smallest element in the final group is expressed as D_a, where D_a equals 4 in the above example. The D_a number is a measure for the load balancing over the available of processors.

As the static output cone partitioning is used in the parallel test system, each processor is assigned the netlist of the output cones associated with it. Consequently, only the circuit portions belonging to more than one output cone need to be replicated. Also, as previously mentioned in chapter 3, the dynamic lists generated by the proposed algorithm is very small and consumes smaller memory segments compared with circuit sizes tested.

4.4 System Modeling

As mentioned earlier, the partition algorithm has a significant impact on the parallel test system performance. We have assumed that the static partitioning algorithm used in the parallel test system distributes balanced loads to the available processors. Consequently, the communication factor between the processors is zero. The idea is to give each processor a version of the test generation algorithm and a portion of a circuit. Each circuit portion represents at least one output cone. Each processor generates a separate test set for the output cone(s) associated with it. With all test sets combined together, a final test set is generated.

Figure 4.2 shows the overall parallel test procedure. The partition algorithm is first initiated to create the necessary load for each processor. Then, each processor is assigned a circuit partition of output cones by receiving the node numbers of the corresponding primary outputs as an initial node list. Each processor then calls the

```
Input   : Circuit netlist.
        : Number of gates, number of PI's, and number of PO's.
        : Number of processors available (N_p).
Output : Test set.

Procedure paral_test_gen( ) {
   partition( );
   assign a balanced load to each processor;
   for each processor's assignment {
      test_gen( );
      /* Generate a test set for each processor's assignment. */
      save the test set in a common area for all processors;
   }
   unique(test set);
   for each vector in the final set {
      fault_sim( );
   }
   compute the fault coverage.
}
```

Figure 4.2. The parallel testing algorithm.

test generation algorithm to generate a test set for this particular circuit partition. As the processor with the largest load completes generating the last test set, a final test set is created by grouping all other sets together. The final set is then minimized by removing all redundant vectors in the set keeping only one version of each vector. A fault simulation program is then run in order to evaluate the test set and calculate the fault coverage.

As may be seen from the above discussions, a real parallel machine is not really needed for implementing the parallel test system. Instead, we have used a single processor machine to implement such a system. The partition algorithm is first run, then, each processor assignment is run sequentially. Of course, this would not be feasible if the partition algorithm uses a dynamic method where process communications is required.

4.5 Summary

This chapter addressed the issues involved in providing a parallel test generation system environment by incorporating the test generation algorithm described in chapter 3 as the main test generation resource. Requirements of an efficient fault partitioning scheme were discussed and a static partitioning scheme is presented. The advantage of using static partitioning scheme has been identified, since it cuts off communication overheads among processors.

CHAPTER 5

EXPERIMENTAL RESULTS

5.1 Introduction

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. In evaluating our test system, we have used the ISCAS'85 [4] combinational benchmark circuits. Table 5.1 shows the eight circuit examples used in this research work. At the time of testing, the circuits are represented in their netlist format.

For the purpose of fault simulation, we have built a single-pattern single-fault simulator. The fault simulator used in our system simulates each node in the circuit for a possible s-a-0 and s-a-1 fault. With each vector simulated, all faults detected by this vector are automatically removed from the fault list. Consequently, after a few cycles of simulation, the process speeds up until the test set is exhausted and only faults which are not detected remain in the fault list. Some of these faults are not modeled in the benchmarks, hence, we compared the output fault list with the modeled faults in order to get the exact fault coverage.

Finally, the proposed test generator and the fault simulator were implemented in the C programming language on a SUN SPARC 1+ workstation. The ISCAS'85 [4] combinational benchmark circuits were used to evaluate the test generation system.

Table 5.1. ISCAS'85 benchmark circuit characteristics.

Circuit name	Function	Total gates	No. of PIs	No. of POs	No. of faults
C880	ALU and control	383	60	26	942
C1355	ECAT	546	41	32	1574
C1908	ECAT	876	33	25	1879
C2670	ALU and control	1193	233	140	2747
C3540	ALU and control	1669	50	22	3428
C5315	ALU and selector	2307	178	123	5350
C6288	16-bit Multiplier	2416	32	32	7744
C7552	ALU and control	3512	207	108	7550

5.2 Evaluation of the proposed test pattern generator

The test pattern generator described in chapter 3 can be configured to generate tests starting with all the primary outputs grouped together. We will refer to this scheme as the grouped output configuration. As will be shown experimentally, the grouped output configuration has a better timing performance than single cone configuration where a set of vectors is generated for each output cone individually as described earlier. However, the size of dynamic lists in the grouped output configuration far exceeds that of the single cone scheme. The number of assignments per node are therefore limited to avoid excessive memory utilization by the grouped output scheme. The same technique was also used in the single cone configuration. The results have shown a further efficiency improvement in terms of computing time and memory space.

The test pattern generator has been evaluated for the two possible configurations. Table 5.2 shows the execution time required for netlist translation and test generation when the generator is in the single cone configuration. The table also shows the test length and fault coverage for different values of the parameter K.

In each case, the majority of the modeled faults were detected at small values of K.

Table 5.2. Real Execution Performance of Our Algorithm with the ISCAS Benchmark Combinational Logic Circuits.

Circuit	Faults	Test Vectors			Coverage			Time (sec)		
		K=2	K=10	K=20	K=2	K=10	K=20	K=2	K=10	K=20
C880	942	64	74	78	95.00	98.15	99.93	0.6	0.8	1.0
C1355	1574	100	106	110	94.10	97.79	99.02	0.8	1.4	1.6
C1908	1879	46	68	94	87.00	92.11	98.99	1.0	1.4	1.9
C2670	2747	68	80	92	84.50	89.20	95.41	2.9	5.1	7.9
C3540	3428	68	94	98	86.20	92.16	95.91	2.8	4.3	6.2
C5315	5350	96	102	108	94.00	96.13	98.22	7.7	12.2	17.2
C6288	7744	94	95	96	99.41	99.41	99.54	5.7	7.2	9.2
C7552	7550	84	100	116	93.21	96.16	97.52	13.2	22.3	32.4

It can be seen that the fault coverage increases as we add more sensitizing paths by increasing the value of K. Our algorithm is at least two order of magnitude faster than those of the PODEM implementation of [15] (shown in table 5.3). As the circuit size increases, our algorithm shows a much better time performance compared to PODEM. Hence, our algorithm performs efficiently with large circuits where computational complexity is critical.

Table 5.4 shows the results obtained using our algorithm in the grouped outputs configuration. As has been mentioned earlier, the timing performance of this configuration is better than the single cone scheme. Moreover, the test length is even less than the single cone configuration. This is expected because in this configuration, the interaction between output cones is allowed and the maximum number of assignments K will be shared by the interconnected cones. Since we are comparing both configurations with the same value of K, the limitations on the circuit capability to release more test vectors is apparent in the grouped outputs scheme. As a result, the test coverage is degraded and a larger value for K is required to achieve a higher fault coverage.

Table 5.3. Results obtained using PODEM algorithm.

Circuit	Test Vectors	Coverage	Time (Seconds)
C880	85	100.00	9.05
C1355	145	99.49	38.02
C1908	150	99.73	49.42
C2670	141	95.52	116.81
C3540	206	96.82	235.38
C5315	161	99.20	150.41
C6288	39	99.94	70.81
C7552	251	97.71	437.00

Table 5.4. Results for the grouped outputs configuration, with K = 20.

Circuit	Test Vectors	Coverage	Time (Seconds)
C880	69	91.00	0.8
C1355	18	83.45	1.2
C1908	61	93.61	1.5
C2670	77	89.92	6.4
C3540	87	90.10	5.0
C5315	96	94.22	13.9
C6288	82	95.76	7.3
C7552	102	92.47	25.9

In order to show how both configurations utilize the memory for the storage of the dynamic lists, Table 5.5 shows the maximum size of memory segments used by the test algorithm at different values of K. Because the limitation on the number of assignments applies to all nodes in a circuit, the size of the dynamic lists linearly changes with the value of K for each configuration. On the other hand, the ratio of memory usage by the grouped outputs scheme to that of the single cone, referred to as the memory reduction, is only a function of the circuit itself. If the circuit complexity is homogeneously distributed over output cones, the ratio will be very high. For example, circuit C6288 is a multiplier, which is characterized by a regular design, has a ratio of memory reduction 33.2 at K equals 20. The other extreme is exhibited by circuits that have poor complexity distribution over the output cones. An example of this circuit is C1355 with a reduction ratio of only 1.18 at K equals 20. The overall memory utilization is thus in favor of the single cone configuration.

Table 5.5. Memory utilization comparison results for the two algorithm configurations.

Circuit	Single cone scheme			grouped outputs scheme		
	K=2	K=10	K=20	K=2	K=10	K=20
C880	0.025k	0.072k	0.083k	0.109k	0.412k	0.591k
C1355	0.105k	0.564k	1.085k	0.128k	1.720k	1.280k
C1908	0.088k	0.473k	0.876k	0.159k	0.764k	1.710k
C2670	0.970k	0.353k	0.508k	0.168k	0.905k	1.819k
C3540	0.143k	0.640k	1.776k	0.656k	2.192k	2.222k
C5315	0.080k	0.373k	0.542k	0.522k	1.787k	2.518k
C6288	0.007k	0.047k	0.102k	0.243k	1.473k	3.388k
C7552	0.310k	0.984k	1.329k	1.315k	3.749k	6.217k

From Tables 5.2 to 5.5, the performance of our algorithm can be evaluated as follows:

- The proposed algorithm outperforms any existing deterministic test pattern gen-

eration algorithms in terms of time complexity and test length.

- The fault coverage is slightly lower than other ATG algorithms, like PODEM and FAN.
- As the circuit size increases, the algorithm shows an efficient performance at low values of K.
- The grouped outputs configuration is more likely to be used in a small and medium size circuit complexity where memory utilization is not critical.
- The single cone configuration is very efficient in generating tests for large size circuits, without compromising the memory utilization.
- The proposed algorithm is very useful if employed as a first phase in a test generation system, replacing the traditional random test generation techniques which has a limited fault coverage and longer test length.

5.3 Static circuit partitioning

The single cone scheme has been chosen for implementation in a parallel test system. As stated above, the static partitioning was carried out by giving each processor an equal number of node assignments. Table 5.6 shows the time spent in partitioning the circuits into groups and distributing the loads on the processors. The optimal number of node assignments G_opt is the average number of assignments, which when given to each processor, results in an exact match of timing performance of all processors. D_a is the maximum load unbalance between two processors expressed as the difference of node assignment numbers. In general, the time taken for circuit partitioning is negligible for the entire benchmarks.

Some of the circuits have been partitioned efficiently, as in C880 and C6288, while other circuits were poorly partitioned as in the case of C1908 and C2670. This raises the design issue as mentioned before. We recall here that the limitations on the number of I/O pins for a VLSI circuit does contribute to the complexity of the testing problem. Adding more pins to a VLSI chip will not resolve the test complexity unless an adequate distribution of logic circuitry among the I/O pins is accomplished. To elaborate, let us take the circuits C2670 and C6288 as examples. Circuit C2670 has 233 input pins and 140 output pins, while circuit C6288 has only 32 input pins and 32 output pins. During the partition algorithm run, we found that more than 80% of logic assignments in the C2670 circuit belonged to only a few output cones, while the other primary output nodes were connected to only one or two gates. As a result, a large portion of the C2670 was hard to test despite the relatively large number of I/O pins in the circuit. Consequently, as shown in Tables 5.2 and 5.3, the fault coverage is low and 4.48% of the modeled faults were undetectable. On the other hand, in the C6288 circuit, only small values of D_a were obtained and circuit complexity was homogeneously distributed among primary output nodes. With only 32 output nodes and double the number of gates compared to C2670, a very high fault coverage was achieved.

Table 5.7 indicates the results obtained from the parallel test system for different number of processors, at K equals 40. The time shown in this table represents the time spent by the processor with the maximum number of node assignment. The test generation time difference between the processor with the maximum number of nodes and that with the minimum, was negligible.

As the test generation time becomes smaller, the time taken for netlist translation and circuit partitioning will be comparable to the test generation time. For example,

Table 5.6. The load distribution over the processors and the time spent in the static partitioning procedure.

Circuit	$N_p = 2$		$N_p = 4$		$N_p = 8$		$N_p = 16$		Time (sec)
	G_opt	D_a	G_opt	D_a	G_opt	D_a	G_opt	D_a	
C880	385	0	179	3	89	18	44	5	0.05
C1355	584	0	292	0	146	0	73	0	0.08
C1908	506	15	253	23	126	32	63	30	0.10
C2670	6852	0	3426	100	1713	103	856	107	0.15
C3540	378	1	189	11	94	23	47	12	0.19
C5315	8462	9	4231	33	2115	54	1057	61	0.29
C6288	1317	5	658	13	329	11	164	10	0.35
C7552	5249	1	2524	20	1312	79	656	90	0.42

Table 5.7. The run time for the parallel test system with different number of processors, for K = 40.

Circuit	Time (seconds)				
	$N_p = 1$	$N_p = 2$	$N_p = 4$	$N_p = 8$	$N_p = 16$
C880	3.70	1.90	0.95	0.6	0.35
C1355	3.60	2.00	1.50	1.20	1.10
C1908	3.70	1.85	1.60	1.50	1.30
C2670	24.50	12.70	9.50	5.90	3.10
C3540	8.90	4.95	4.82	3.70	3.50
C5315	62.70	28.25	17.11	12.5	8.90
C6288	15.75	9.00	8.25	7.35	6.50
C7552	64.55	33.50	22.10	18.00	10.1

the circuit C7552 takes 6.1 seconds for netlist translation and partitioning, while the test generation takes approximately 26 seconds (for K = 20, Table 5.1). This ultimately puts a limit on the extent of speedup especially for a large number of processors. The speedup figures can be extracted from Table 5.7, at a value of K being 40. The results shown in Table 5.7 indicate that for large circuits, we can get fairly high speedup ratios. The lower limit on overall test time is the netlist translation and circuit partitioning time. For example, as shown in Table 5.7 with 16 processors, the circuit C7552 requires only 4 seconds for test generation while the other 6.1 seconds were spent in netlist translation and circuit partitioning phase. The actual speedup ratio would be (64.55-6.1)/(10.1-6.1) = 14.6, approaching the number of processors.

The fault coverage for the parallel test system matches the single processor test generation system with the same value of K. For instance, the fault coverage for the parallel test system with K equals 20 matches that shown in Table 5.2. This is because the test generation algorithm inherently partitions the circuit into output cones. In the parallel test system, the same job was done by first partitioning the circuit into output cones and then distributing the jobs among processors.

5.4 Summary

In this chapter, the implementation of the algorithm presented in chapter 3 was discussed. Experimental results on large circuits show that our algorithm outperforms other existing test generation algorithms. The new algorithm is presented in two different configurations. Advantages and disadvantages of each configuration have been investigated. The single cone scheme has been incorporated into a parallel test generation system. Also, a static load partitioning method has been used efficiently in balancing loads over the available processors. Finally, experimental results based

on an implementation of our algorithm in a parallel system model on SUN SPARC1+ work-station were presented. Results have shown that considerable speedup factors were realized due to the efficiency of the test generation algorithm. Memory utilization has also been shown to be very small compared to the circuit size. The overall test system has yielded a high fault coverage and provides time efficient procedures to generate tests for large size combinational circuits.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

The rapid advances in integrated circuit technology have made possible the fabrication of digital circuits with a very large number of devices on a single chip. This complexity is coupled with an increase in the ratio of logic to pins which drastically reduce the controllability and observability of the logic on the chip. In addition, there are new and subtle failures being observed with VLSI circuits. These problems are already causing difficulties with the testing of the existing complex chips. Testing is expected to become even more difficult with the higher complexity chips that are being proposed. This research work proposes a new technique for designing test generation algorithms with better time complexity than the existing ones.

In this research, we described a variety of fault and error models which are used as the basis for designing fault-tolerant VLSI systems. The fault models describe physical defects and failures and the input patterns which will expose them, and are suitable for testing. Error models on the other hand describe the effects on the functional outputs of defects and are useful for on-line error detection. The models are described at the gate level of abstraction.

The test generation problem has also been presented as a space search for test patterns which detect single stuck-at faults at the gate level of abstraction. It has been shown that the test generation problem is a complex problem and is considered to be

NP-complete. Different approaches have been used to tackle the test problem, either by randomly generating test vectors or by using other deterministic test generation methods. The test generation as a space search problem has evolved in designing efficient algorithms for test generation such as PODEM and FAN. Most of the test systems reported for this decade are based on these two algorithms.

The current test generation algorithms can generate test vectors for complex combinational circuits and guarantee 100 percent coverage for testable faults. However, the test generation time increases exponentially with the increasing circuit complexity. A new approach which combines simplicity and fast performance has been developed for the test generation of large combinational circuits. The similarities of this algorithm with current approaches have been identified. The test generation problem has been formulated as a global search of sensitizing paths between the input and the output primary nodes. To achieve this objective, a circuit environment is created such that all faults which can be detected by multiple - path sensitization were detected. The algorithm may not detect all faults that are sensitized with only a single path. Hence, the fault coverage limitation will only be a function of the circuit complexity. The fact that the algorithm does not process a decision tree makes it fast enough to be compared with random test algorithms.

The issues involved in providing a parallel test generation system environment by incorporating the proposed test generation algorithm as the main test generation resource have been discussed. Requirements of an efficient fault partitioning scheme were discussed and a static partitioning scheme was presented. The advantage of using a static partitioning scheme has been identified by cutting off communication time overhead among processors.

The implementation of the test generation algorithm is presented with experimental results derived from the ISCAS'85 combinational benchmark logic circuits. The results on large circuits suggest that our algorithm outperforms the other test generation algorithms. The new algorithm is presented in two different configurations. Advantages and disadvantages of each configuration have been investigated. The single cone scheme has been incorporated into a parallel test generation system. Also, a static load partitioning method is efficiently used in balancing loads over the available processors. Finally, experimental results based on an implementation of our algorithm in a parallel system model on a SUN SPARC1+ work-station were presented. Results have shown that considerable speedup factors were realized due to the efficiency of the test generation algorithm. Memory utilization has also been shown to be very small compared to the circuit size. The overall test system has yielded a high fault coverage and provided time efficient procedures to generate tests for large size combinational circuits.

We believe that our algorithm can efficiently replace the random test generators used as a first phase in test systems. Although our algorithm runs at a comparable speed to random test generators (RTG), the fault coverage is much higher than the RTG techniques. Consequently, if our algorithm is integrated with other deterministic test pattern algorithms, like PODEM, a very efficient test system will result. PODEM, for instance, will search for test vectors for a very small test set which will save on overall test time. We hope that this system can be integrated in the future.

REFERENCES

[1] P. Agrawal and V. D. Agrawal. Probabilistic Analysis of Random Test Generation Method for Irredundant Combinational Logic Networks. *IEEE Trans. Comp.*, C-24(7):691–695, July 1975.

[2] V. D. Agrawal and M. R. Mercer. Testability measures - what do they tell us? *IEEE Test Conf., Chirry Hill, Phil.*, pages 391–396, 1982.

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

[4] F. Brglez and H. Fujiwara. A neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in FORTRAN. *IEEE International Symposium on Circuits and Systems*, June 1985.

[5] H. Fujiwara and T. Shimono. On the Acceleration of Test GenerationAlgorithms. *IEEE Trans. Comp.*, C-32:1137–1144, Dec. 1983.

[6] M. Garey and D. Johnson. *Computer and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, San Francisco, 1987.

[7] P. Goel. Test Generation Costs Analysis and Projections. *Proc. 17th Design Automation Conference*, pages 77–84, June 1980.

[8] P. Goel. An Implicit Enumeration Algorithm to Generate Tests For Combinational Logic Circuits,. *IEEE Trans. Comp.*, C-30:215–222, March 1981.

[9] L. H. Goldstien and E. L. Thigpen. Scoap: Sandia controllability/observability analysis program. *Des. Aut. Conf., Minneapolis, Minn.*, June 1980.

[10] E. Horowitz and S. Sahni. *Fundamentals of Computer Algorithms.* Computer Science Press, Washington, DC, 1978.

[11] O. H. Ibarra and S. K. Sahni. Polynomially Complete Fault Detection Problems. *IEEE Trans. Comp.*, C-24:242–249, March 1975.

[12] E. J. McCluskey and F. W. Clegg. Fault Equivalence in Combinational Logic Networks. *IEEE Trans. on Comp.*, C-20:1286–1293, Nov. 1971.

[13] E. J. McCluskey, S. Makar, S. Mourad, and K. D. Wagner. Probability Models for Pseudorandom Test Sequences. *IEEE Trans. on CAD*, 7(1), Jan. 1988.

[14] Hyoung B. Min and William A. Rogers. Search Strategy Switching : An Alternative to Increased Backtracking. *Int. Conf. on Testing*, Sept. 1989.

[15] Srinivas Patil and P. Banerjee. Performance Trade-Offs in a Parallel Test Generation/Fault Simulation Environment. *IEEE Trans. on CAD*, Dec. 1991.

[16] D. K. Pradhan. *Fault-Tolerant Computing.* Prentice Hall, 1986.

[17] J. P. Roth. Diagnosis of Automata Failures, A Calculus and A Method. *IBM J. Res. Dev.*, 10:278–291, July 1966.

[18] D. R. Schertz and G. Metze. A New Representaion for Faults in Combinational Digital Circuits. *IEEE Trans. on Comp.*, C-21:858–866, August 1972.

[19] Michael M. Schulz, Erwin Trischler, and Thomas M. Sarfert. SOCRATES: A Highly Efficient Test Pattern Generation System. *IEEE Trans. On CAD*, 7(1), Jan. 1988.

[20] J. J. Shedletsky and E. J. McCluskey. The Error Latency of A Fault in A Sequential Digital Circuit. *IEEE Trans. Comp.*, C-25:655–659, June 1976.

[21] J. A. Waicukauski, E. A. Eichelberger, D. O. Forlenza, E. Lindbloom, and T. McCarthy. Fault Simulation for Structured VLSI. *VLSI Systems Design*, page 20, Dec. 1985.

[22] A. Yousif and Jun Gu. An Efficient Global Search Algorithm for Test Generation. *Submitted to IEEE Trans. On CAD for publication*, 1992.