CH2PQ
UP 8
1989
S784

UNIVERSITE DE MONTREAL

DETECTION OF HARD FAULTS IN COMBINATIONAL LOGIC CIRCUITS

par

David H. STANNARD
DEPARTEMENT DE GENIE ELECTRIQUE
ECOLE POLYTECHNIQUE

MEMOIRE PRESENTE EN VUE DE L'OBTENTION
DU GRADE DE MAITRE ES SCIENCES APPLIQUEES (M.Sc.A.)

DECEMBRE 1989

© David H. Stannard 1989

National Library of Canada
Bibliothèque nationale du Canada

Canadian Theses Service
Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-58199-9

Canada

UNIVERSITE DE MONTREAL

ECOLE POLYTECHNIQUE

Ce mémoire intitulé

DETECTION OF HARD FAULTS IN COMBINATIONAL LOGIC CIRCUITS

est présenté par: David H. Stannard

en vue de l' obtention du grade de: Maitre es sciences appliquées (M.Sc.A.)

M. Bernard Lanctôt, Président, ing.

M. Jean-Charles Bernard, Ph.D.

Mme. Bozena Kaminska, Ph.D.

M. Yvon Savaria, Ph.D.

ABSTRACT

Integrated circuits have become increasingly complex while their users have become more stringent in their quality requirements and simultaneously exhibiting a larger appetite for newer products at a faster rate. Indeed, some integrated circuits resemble the statement of *systems on silicon* and are being designed by systems designers instead of the traditional IC designer.

When one adds to these tough specifications the need for high reliability, the ability to be manufacturable, and the desire to provide a product ready for volume delivery within six months of defining of the circuit's objective specification, it becomes apparent that the current testing bottleneck can no longer be tolerated. It is desirable to provide a design environment which will allow the design team to identify testing difficulties such as those logic faults, due to circuit fabrication, which are hard to test. It is critical that these faults are identified as early in the design as possible, and that the necessary information describing their origins be provided to the design team thus allowing the appropriate design corrections to be made given knowledge of the design's constraints: cost, circuit performance, yield at the wafer level, test time, and time to market; all of these are known from the original objective specifications which form a contract on all the parties involved in the design process.

This thesis examines an approach to defining and indentifying hard to test faults contained within a combinational logic circuit based upon the use of cost constraints imposed upon the automatic test pattern generation (ATPG) process. This approach was used since past heuristic-based methods such as testability measures have not provided an accurate means of detecting the hard faults' presence. By coupling the concept of cost constraints to a variant of an efficient and extensible automatic test pattern generation tool (Fujiwara's FAN), it was hoped that one of the heuristic's past problems, the lack of coupling to the test generation effort, could be corrected.

In addition to creating this tool, a new circuit modelling method was developed: *graph binary decision diagrams* (gBDD) which was created to provide a

means of including information about the circuit's functionality and structural aspects. A simple fault simulator, designed around the single stuck at fault model, was incorporated into the hard fault identifier *HUB*. HUB also provides feedback indicating the causes of the hard faults by indicating which ATPG phase is the most costly (according to time measurements), which ATPG phases cause backtracks to occur, and which logic elements can be clearly defined as the originators of the backtracks.

The results of this research have indicated that this approach has merit, but a problem is apparent: without circuit schematics for the circuit being evaluated, it is difficult to verify the preciseness of HUB's results and to allow meaningful analysis of the circuit. However, for the circuits evaluated and for which schematics were available, HUB provided information about the reasons for the redundant logic faults (not considered as hard faults in this thesis) identified by exhaustive searches of the circuit whose correctness was verified by comparision with the circuit's schematics. The cost information also shows the relative costs of the various ATPG phases and clearly identified that the majority of the backtracks are caused by the net justification phase.

The most significant result of this research is the creation of the new circuit modelling technique *gBDD* that allows the vertical integration of a wide variety of previous researchers' works in manipulating functional related circuit information and structural based circuit data. The ability to generate functional based tests (as opposed to tests built around a fault model such as the traditional stuck at fault) is made possible by the functional circuit information and is derived from the binary decision diagram portion of the *gBDD* circuit model. The manipulation of graph portion of the new circuit model, extracted from the circuit's structure, is useful for determining the placement of diagnostic test points derived from knowledge about the circuit's reconvergent fanout characteristics and for calculating testability metrics in hierarchically described circuits.

SOMMAIRE

Une nouvelle méthode a été créée afin de découvrir la présence des défauts dans un circuit logique, combinatoire seulement, qui sont difficiles à détecter (HFs) avec des essais pour la vérification du circuit. On a proposé de les classer et de faire leur détection en utilisant des heuristiques fondées sur les coûts et les budgets de vérification pour le circuit pendant qu'un algorithme tente de générer des tests pour les défauts choisis. Cette méthode a été choisie afin de permettre l'intégration des contraintes de ressources, les heuristiques qui découlent, les vrais efforts requis pour la génération des vecteurs, et de ramasser des informations sur les causes de ces HFs. Grâce à l'algorithme de génération automatique des vecteurs de test FAN, qui sert comme fondation pour HUB (le système pour découvrir la présence des HFs), on peut même détecter s'il y a des HFs créés par la redondance lors de la conception du circuit.

Pour supporter HUB, une nouvelle méthode de modélisation du circuit a été créée - gBDD - (graph binary decision diagram): cette modélisation permet de combiner les avantages des informations structurelles, décrites par les graphes, ave les informations de comportement détaillées par les BDDs (binary decision diagrams).

HUB est un ensemble de logiciels qui permet de traduire une description du circuit en modélisation gBDD, qui génère des mesures afin de les utiliser avec des heuristiques pendant la phase de génération des vecteurs de test, et qui tente d'identifier la présence de ces défauts difficiles à détecter par l'algorithme de génération des vecteurs. HUB mesure les coûts de cette phase de génération et le logiciel garde ces données dans un fichier pour un traitement ultérieur par

l'usager. Les logiciels ont été faits pour que l'usager puisse modifier les contenus avec les logiciels commerciaux et en utilisant des commandes du système d'exploitation de l'ordinateur, surtout ceux d'UNIX. Donc cette recherche a créé un environnement de travail. L'usager peut se servir d'HUB comme simple générateur de vecteurs ou même pour apprendre comment ce genre de logiciel fonctionne.

HUB a été utilisé pour tenter d'identifier la présence des HFs dans des circuits combinatoires d'ISCAS 1985. Malgré que les principes de base permettent de découvrir et de définir ces défauts, selon la définition de coût et en absence des retours en arrière (en anglais, *backtracks*), HUB ne génère pas assez de données pour bien identifier les causes de ces HFs. Les résultats indiquent que HUB peut identifier les raisons pour les retours en arrière par l'identification du noeud responsable pour ce retour et que la majorité de ces retours sont causés par la phase de justification des noeuds à l'intérieur du circuit. HUB a réussi à mesurer les coûts de chaque phase majeure de la génération automatique des vecteurs de test malgré que le système d'UNIX semble avoir un problème à bien mesurer la quantité de temps. Pour permettre la future correction du circuit, HUB imprime des informations sur les heuristiques responsables des difficultés pendant les phases majeures de la générations de ces vecteurs.

Par contre, le modèle du circuit (gBDD) a fait ses preuves comme méthode pour aider la génération des vecteurs de test et est devenu le résultat le plus important de cette recherche. Ce modèle a permis la modification de la façon de générer les vecteurs de test et d'ajouter des nouvelles heuristiques. Ce modèle permet l'utilisation d'autres modèles de défauts avec des modifications au logiciel.

NOTES of THANKS

During this period of research, many people have been instrumental in providing technical support, guidance, and the use of tools that would otherwise not be available:

A special note of thanks to my supervisor and mentor, Mme. Bozena Kaminska, for her support and efforts during the past two years. She has allowed me to experiment with ideas and concepts that were foreign to me before embarking down these paths and caused me to expand my horizons.

My wife and children also deserve an enormous recognition for their support and encouragement during the hectic times of university, full time employement, and the daily requirements of being a father in a family unit.

My employer, MITEL S.C.C. has graciously provided me with access to a multitude of computer and software tools, absences from my employment, and encouragement to forge ahead in the wonderful domain of integrated circuit technology.

To all the professors, administrative staff too numerous to list; you all know who you are - I thank you too.

Table of Contents

Abstract iv

Sommaire vi

Notes of Thanks viii

List of Tables xii

List of Figures xiv

List of Graphs xvii

List of Acronyms xviii

Table of Appendices xix

1.0 Introduction. 1

2.0 Previous Work in Identifying Hard to Test Faults (HFs). 5

2.1 The effect of reconvergent fanout and redundancy. 5

2.2 Testability Measures (TMs). 8

2.2.1 SCOAP - Sandia Controllability Observability Analysis Program. 11

2.2.2 COP - Controllability Observability Program. 14

2.3 Using of ATPGs to detect HFs. 17

2.4 Previous use of cost in Testability Analysis. 20

2.4.1 Breuer's Sensitivity Functions. 20

2.4.2 Random Test Cost Functions. 21

2.4.3 Sequential Test Cost Functions. 23

2.4.4 Summary of Cost Functions. 24

3.0 Review of Automatic Test Pattern Generation (ATPG). 26

3.1 Fault modelling. 26

3.1.1 Single stuck at fault model and 5 value logic. 27

3.2 Single versus Multiple path sensitization. 31

3.3 The Four ATPG phases of Deterministic Gate Level Test Generation. 31

3.4 Random Test Pattern Generation and Hybrid Methods. 36

3.5 Review of the FAN algorithm. 39

3.6 Backtrack reduction methods and the importance of heuristics. 43

3.7 Summary. 44

4.0 Mixed Graph - Binary Decision Diagram (gBDD) Circuit Model. 45

4.1 Introduction. 45

4.2 A Review of graph techniques. 46

4.3 A Review of binary decision diagrams (BDDs) techniques..... 49

4.4 gBDD - Graph Binary Decision Diagrams. 54

4.5 Summary. 58

5.0 Detection of Hard Faults using HUB. 59

5.1 Introduction to budgetary constraints..... 59

5.2 The HUB algorithm. 64

5.3 Important HUB attributes. 71

5.3.1 ATPG control by Graph Node (GN)..... 72

5.3.2 pdcf Personality File. 74
5.3.3 Fault model, fault collapsing and simulation. 75
5.3.4 Interactive Mode and Information Feedback. 78
5.3.5 Other HUB features. 80
5.3.6 Hierarchy Provisions. 81
5.3.7 Summary. 82
6.0 Results. 83
6.1 Circuit characteristics of used for results. 85
6.2 Comparison of gBDD. 85
6.2.1 gBDD Temporal performance. 85
6.2.2 File sizes. 88
6.3 ATPG related results. 90
6.4 Fault Simulation related results. 91
6.5 Hard Fault Detection. 93
6.5.1 Extended example using Image. 93
6.5.2 General Results. 107
6.6 Summary. 112
7.0 Conclusions. 114
8.0 Bibliography. 117

List of Tables

Table 2.2.1.1: SCOAP initial conditions. 13

Table 2.1.1.2: SCOAP values for the full adder circuit. 15

Table 2.2.2.1: COP initial conditions. 16

Table 2.2.2.2: COP values for the full adder circuit. 19

Table 3.1.1.1: Rules for Difference intersection. 30

Table 6.1.1: ISCAS and other test circuit characteristics. 86

Table 6.2.1.1: gBDD time distributions 88

Table 6.2.2.1: Circuit Model File Requirements (k bytes). 89

Table 6.3.1: Partial ATPG results - no simulation 90

Table 6.4.1: ATPG results with simulation vs fault simulation. 92

Table 6.5.1.1: Cumulative total cost distribution. 104

Table 6.5.1.2: Cumulative propagation distribution. 105

Table 6.5.1.3: Cumulative justification cost distribution. 105

Table 6.5.2.1: Justification backtrack information. 108

Table 6.5.2.2: Mean of backtracks. 109

Table 6.5.2.3: Mean and standard deviation of total costs. 109

Table 6.5.2.4: Mean and standard deviation of propagation total costs. 110

Table 6.5.2.5: Mean and standard deviation of justification total costs. 110

Table 6.5.2.6: Mean and standard deviation of backtrack total costs. 111

Table 6.5.2.7: Mean and standard deviation of backtrace total costs. 111

Table 6.5.2.8: Mean and standard deviation of implication total costs. 112

List of Figures

Figure 2.1.1: Example of reconvergent fanout. 7

Figure 2.1.2a: Graph with reconvergent fanout. 8

Figure 2.1.2b: Graph without reconvergent fanout..... 8

Figure 2.2.1: Blocking condition example. 10

Figure 2.2.2: Line justification example. 11

Figure 2.2.1.1: SCOAP pseudo code. 12

Figure 2.2.1.2: Some elements and their SCOAP relations..... 13

Figure 2.2.1.3: Full adder circuit. 14

Figure 2.2.2.1: COP pseudo code..... 16

Figure 2.2.2.2: Some elements and their COP relations..... 17

Figure 2.4.2.1: ESPRIT pseudo code..... 21

Figure 2.4.2.2: Fault Coverage function. 23

Figure 3.1.1a: CMOS inverter with output shorted to VDD. 28

Figure 3.1.1b: CMOS inverter with output shorted to VDD. 28

Figure 3.1.1.1: CMOS inverter with output stuck at 1..... 29

Figure 3.2.1: Path sensitization example..... 32

Figure 3.3.1: ATPG example..... 33

Figure 3.3.2: Error propagation example..... 34

Figure 3.3.3: Net justification example. 35

Figure 3.3.4: A typical ATPG algorithm. 36

Figure 3.4.1: A typical LFSR circuit. 37

Figure 3.4.2: Random Test Pattern Generation..... 38

Figure 3.5.1: Types of nets in FAN..... 41

Figure 3.5.2: FAN pseudo code..... 42

Figure 4.2.1: Full adder..... 47

Figure 4.2.2: Full adder's graph..... 48

Figure 4.2.3: Full adder's relation matrix..... 50

Figure 4.3.1: n - input AND gate's BDD..... 51

Figure 4.3.2: Activated modulo 2 BDD..... 52

Figure 4.3.3: Typical BDDs..... 53

Figure 4.4.1: Modified BDD for AND gate..... 56

Figure 4.4.2: gBDD for full adder circuit..... 57

Figure 4.4.3: Graph node and mBDD node data structures..... 58

Figure 5.1.1: Design phases..... 62

Figure 5.1.2: Budget importance..... 63

Figure 5.1.3: Budget formulae..... 64

Figure 5.2.1: Creation of gBDD circuit..... 65

Figure 5.2.2: HUB's inputs and outputs..... 66

Figure 5.2.3: Fault coverage curve..... 67

Figure 5.2.4: HUB's pseudo code..... 68

Figure 5.2.5: HUB ATPG's pseudo code..... 69

Figure 5.2.6: HUB Cost Accounting Data Structure..... 70

Figure 5.3.1.1: HUB's graph node pointer list..... 73

Figure 5.3.2.1: 2-input AND gate's pdcf personality file..... 74

Figure 5.3.3.1: Fault collapsing..... 76

Figure 5.3.3.2: Fault simulation using lists..... 77

Figure 5.3.3.3: Fault simulation sample output data..... 79

Figure 6.5.1.1: Image circuit subsection 95

Figure 6.5.1.2: Partial HUB hard fault data..... 107

Figure A.1: HUB's options and syntax. 124

Figure A.2: HUB's data output. 125

Figure A.3b: HUB's SILOS II' vector output..... 126

Figure A.3a: HUB's documented vector output..... 126

Figure A.4: Neutral netlist syntax. 127

Figure A.5: Permissible element list..... 127

Figure A.6: Full adder netlist example. 128

Figure B.1: Listing of Image's graph node output names..... 129

Figure B.2: Example of HUB's accounting data..... 130

List of Graphs

Graph 6.2.1.1: gBDD file storage time. 87

Graph 6.5.1.1: Primary Backtrack Phases. 96

Graph 6.5.1.2: Secondary Backtrack Phases. 96

Graph 6.5.1.3: Graph Node causing Backtrack Phase. 97

Graph 6.5.1.4: Graph node number (reduced fault dictionary). 98

Graph 6.5.1.5: Propagation phase cost histogram. 99

Graph 6.5.1.6: Justification phase cost istogram. 99

Graph 6.5.1.7: Backtrace phase cost histogram. 100

Graph 6.5.1.8: Backtrack phase cost histogram. 100

Graph 6.5.1.9: Implication phase cost histogram. 101

Graph 6.5.1.10: Total cost histogram. 101

Graph 6.5.1.11: Budget histogram. 102

Graph 6.5.1.12: Total cost histogram (fault simulation). 102

Graph C.1: C17 total cost histogram. 131

Graph C.3: C880 total cost histogram. 132

Graph C.2: C95 total cost histogram. 132

Graph C.5: C2670 total cost histogram. 133

Graph C.4: C1908 total cost histogram. 133

Graph C.6: Full adder total cost histogram. 134

List of Acronyms

ATPG	Automatic Test Pattern Generation.
BDD	Binary Decision Diagram.
COP	Controllability Observability Program.
CDP_j	Cummulative detection probability.
D	Difference.
FAN	FANout oriented ATPG.
FCE	Fault Coverage Estimate.
gBDD	graph BDD.
HUB	Hard fault detection Using Budgetary constraints.
HF	Hard Faults.
mBDD	modified BDD.
pdcf	primitive d cube of failure.
Pd_{li}	Probability of detecting net l stuck at i.
PI	Primary Input.
PO	Primary Output.
PODEM	Path Oriented DEcisioin Making.
SCOAP	Sandia Controllability Observability Analysis Program.
TM	Testability Measure.
UUT	Unit Under Test.
UUT_PI	UUT's Primary Input.
UUT_PO	UUT's Primary Output.
VICTOR	Vlsi Identifier of Controllabillity, Testability, Observability and Redundancy.

List of Appendices

9.0 Appendix A..... 124

10.0 Appendix B..... 129

11.0 Appendix C..... 131

1.0 Introduction.

The continued increase in integrated circuit (IC) complexity, the tremendous pressure on reducing the time to market for these dense products coupled with the tumultuous drive to design products having inherently better quality and manufacturability with the first product revision has resulted - once again - in a heightened awareness of the testing bottleneck. Test and design engineers understand from their first hand experience, the conflicting interests caused by the requirements of high quality (expressed as a test metric), short test application times, the need for high yields - an ambiguously defined term - and reduced test costs.

The literature mentions that high test quality, such as 95% *fault coverage metric* using the *single stuck at fault model*, provides for more reliable products [McCluskey 88]. All testing, and hence the ensuing high quality, requires the ability to control the testing stimuli and observe the deterministic results. An IC design's controllability and observability are key aspects that have been addressed through the design methodology and grouped loosely under the monicker of *Design For Test* (DFT). Ad hoc techniques, such as those listed by Bardell [Bardell *et al.* 86], and the more formal DFT ideas, examples are listed in [McCluskey 86][Bardell *et al.* 86], have been suggested as means to approach the testing problem from the hardware perspective. Even more recently, researchers have been working on synthesis which includes the designing of testable hardware as demonstrated in papers from [Devadas *et al.*][Sangio 88a] [Beenker *et al.* 89]. There has been a tendency to use these methods as the proverbial ''silver bullet'' intended to slay the testing monster. The negative effect of some DFT solutions on product yield, die size and even circuit performance has led to its use on an as needed basis in more recent years - as exemplified by the work on *partial scan path* [Agrawal *et al.* 88].

At the 1989 International Test Conference, the invited speaker, Dr. Tom Williams of the IBM corporation stressed the need for quality by stating that achieving the 100 ppm (parts per million) of defective units requires a minimum of 100% *single stuck at*

fault coverage and that the integrated circuits must be tested for delay faults. With the increasing demand of the computationally intense software verification activities such as fault simulation, one must design testable, manufacturable ICs using methods that relieve this tool-related burden [D&T 89] [Miczo 86].

Therein lies one of the fundamental problems facing the testing community - identification of what is hard to test and what is easily tested. Much work in the field of estimating circuit testability, using the concept of *testability measures* (TMs), has met with limited success. Despite the introduction of many different TMs - SCOAP [Goldstein 79], COP [Brglez 85], VICTOR [Ratiu 82] - subsequent authors have shown that there is little confidence in the ability to correlate these static TMs with the reality of testing the circuits; this despite their use in guiding automatic test pattern generation (ATPG) for pruning the algorithm's decision search space. Even Ivanov's work using dynamic TMs [Ivanov 85] provided little extra benefit once the added computational loading was considered. The underlying difficulty appears to be the use of heuristics to create linear algorithms to approximate an NP complete problem.

The use of ATPGs to detect faults which are hard to test (HFs) is a definite possibility since, when an algorithm is employed, an exhaustive search will positively identify the presence of redundant logic and other HFs. ATPGs are known to be NP complete and their indiscriminate use can result in large computer run times. Guidance heuristics, such as TMs and backtrack limits, were introduced as attempts to reduce this excessive computer usage. However, as Fujiwara [Fujiwara 85] and Marlett [Marlett 89] indicate, it is important to reduce backtracks by making the correct decision and thus try to avoid labelling testable faults as HFs due to erroneous decisions. Ivanov suggested that the use of TMs actually causes the ATPG's failure [Ivanov 85] for certain conditions.

ATPGs and TMs suffer another limitation: they indicate that a fault is hard to test but fail to indicate the cause other than the fact that the backtrack limit has been exceeded (ATPG) or that the testability is deteriorating. In order to correct the

underlying problem, it is desirable to understand the HF's origins - is there no error propagation path? - is there a conflict due to the assignment of logic values to the nets?

This thesis proposes to identify the existence of HFs in logic circuits (restricted to combinational logic) based upon some new concepts which will be used to extend the previous work of other researchers. The concept of costs - and the budgetary constraints which arise from the cost of work - will be applied as a true measure of what type of fault is a hard fault. This is the primary direction taken by this paper. Cost accounting techniques will be applied to the various phases of an extended version of a well known ATPG (Fujiwara's FAN [Fujiwara 85]): these phases are the error propagation (difference or D propagation), net justification, the net implication (the simulation due to application of known values to the logic circuit's nets), the decision space creation (backtracing) and the re-evaluation of previous decisions (backtracking).

The concept of cost will be used to tie in design related knowledge - i.e. how much emphasis the designer wishes to place on the thorough testing of a given design - by allowing the designer to specify the total cost constraints for the unit under test (UUT). This will allow the use of an efficient algorithm to perform an exhaustive search in the worst case or unlimited budget, and to also examine the impact of severe budget considerations. Recent estimates indicate that fully one half of a large IC's development cost may be due to the impact of the IC's testing [Henckels 88a] [Henckels 88b].

In order to support this method, new heuristics have been developed and coupled with a new circuit modelling technique. These aid an extended FAN algorithm in arriving at a solution. The novel circuit model provides help in guiding the decision making process and provides for the future use of hierarchical circuit description in order to reduce the test generation time. It also permits the removal of the single stuck at fault model restriction.

This thesis is divided into 7 Chapters and has the following organization:

- Chapter 2.0 reviews previous research on identifying hard to test faults (HFs). Testability Measures (TMs), specifically SCOAP and COP, are discussed; the effect of reconvergence and fanout in circuits, and the previous use of cost in Testability Analysis are reviewed; and the means by which automatic test pattern generation have all been used to attempt to discover the presence of HFs at the various phases of circuit design are described.

- Chapter 3.0 provides a review of the key concepts involved with automatic test pattern generation (ATPG) required to introduce our method of hard fault identification - *HUB*. A summary of the basic single stuck at fault model, the internals of ATPG algorithms (specifically Fujiwara' s FAN since this has served as a basis for HUB), and the basic framework of definitions is undertaken.

- Chapter 4.0 details the mixed graph and binary decision diagram circuit model which is required by HUB. Its ability to describe the circuit's functionality and structure, to provide hierarchical modelling of mixed combinational and sequential circuits, and to allow different fault models to be employed are presented.

- HUB - *Hard fault detection Using Budgetary constraints* - is explained in depth in Chapter 5.0.

- HUB's heuristics, the introduction of the budgetary constraint concept that is central to our HF identification tool, and a preview of how the circuit modelling technique will be closely coupled with to HUB, are described in Chapters 4.0 and 5.0

- The results of measurements obtained from HUB on some of the 1985 ISCAS combinational benchmark circuits are provided in Chapter 6.0; this data is also analysed and explained in this same chapter. Chapter 7.0 provides the conclusions.

2.0 Previous Work in Identifying Hard to Test Faults (HFs).

This chapter will provide insight into the existence of hard to test logic faults caused by the nature of the design methodologies and how various techniques have been employed to attempt the detection of their existence at various stages of the circuit design cycle.

The important effects of reconvergent fanout and logic redundancy in a logic circuit provides a review of some basic definitions and leads into the use of tools; specifically tools used to manipulate this information. Additional heuristic tools, used to estimate the vaguely defined circuit testability, are presented using two traditional testability measure sets: (1) SCOAP; and (2) COP which are linear approximation approaches to the NP complete test generation problem. Subsequently, the use of automatic test pattern generation tools in detecting HFs will be described including additional explications as to how the tool and its circuit environment can cause faults to become hard to test.

A review of sensitivity cost functions, random test cost functions, and sequential test cost functions introduce more fully the concept of applying cost measures in improving a circuit's testability. These sections will also provide insights into the diversity of techniques, metrics, and heuristics employed in testability analysis.

2.1 The effect of reconvergent fanout and redundancy.

The presence of reconvergent fanout definitely causes difficult to test faults - HFs. Bell states that backtracking operations within ATPGs, caused by redundancy and the decision making process, are caused by reconvergent fanout [Bell Taylor 88]. He also quotes Savir as indicating that reconvergent fanout [Savir 83] may cause the inability of TMs to correlate with the difficulty that ATPGs can experience in trying to generate a solution. Reconvergent fanout also prompted Fujiwara to develop the FAN ATPG algorithm [Fujiwara 85]. These HFs have driven several researchers to develop techniques which analyse the circuit's structure and indicate either the

points at which *test points* (used to increase the observability within a circuit) need to be inserted or whether the circuit is difficult to test - TMs.

Russell provided techniques, based upon graph theory, for determining the diagnostic resolution of faults within combinational logic circuits with application to the insertion of test points [Russell Kime 71]. These techniques only examined the circuit's structure without any consideration of the UUT's function or the function of the individual modules from which the UUT is composed. This work was specifically aimed at determining where test points - a structural change - could be inserted. They also wished to evaluate the network structure's contribution to the UUT's fault diagnosis properties. Batni later extended these principles to diagnostic test generation techniques [Batni Kime 76].

Reconvergence in a circuit exists when a signal, emanating from a common point known as the *fanout stem*, flows along more than one path each of which are referred to as *fanout branches,* and then arrive at a common circuit primitive - the *reconvergence point* [Kirkland Mercer 88]. Figure 2.1.1 shows a simple example with the appropriate points annotated. Note that the introduction of reconvergence alters the tree like structure of the circuit's associated graph. Figure 2.1.2a shows the circuit graph in the presence of reconvergent fanout and Figure 2.1.2b demonstrates the effect of replacing the fanout branches with two independent signal sources.

Redundancy can generate HFs also. A *redundant fault* is caused by a *redundant connection* - a connection, which when removed and replaced by a fixed logic value of 0 or 1 - does not alter the output functions of a circuit. A circuit that contains no redundant connections is called *irredundant.* A theorem [Miczo 86] follows from this definition.

THEOREM 2.1: All stuck at 0 and stuck at 1 faults contained in the combinational circuit UUT, are detectable if and only if the circuit is irredundant.

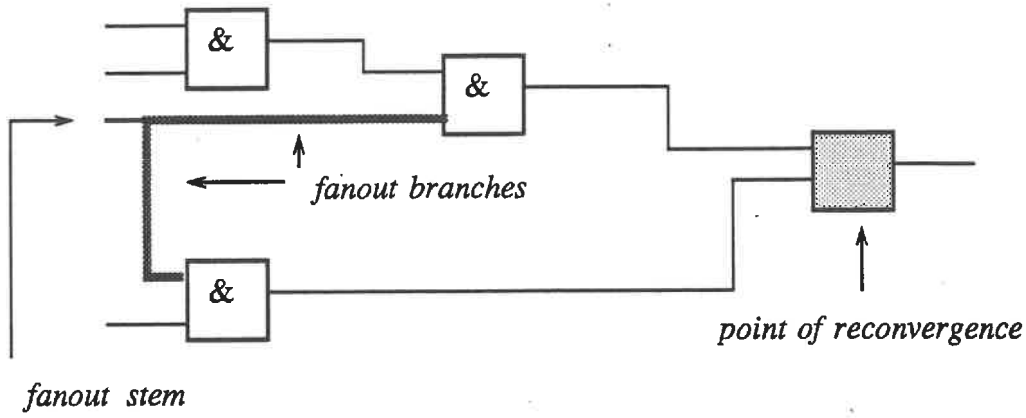


Figure 2.1.1: Example of reconvergent fanout.

[Russell Kime 71] state that *reconvergence is a necessary condition for redundancy*. They extend theorem 2.1 to state the sufficient condition for detecting every fault within an irredundant circuit.

THEOREM 2.2: If a circuit is irredundant, then every stuck at 0 and stuck at 1 fault on a net and at a module's inputs and outputs in the network is detectable. If a network is irredundant, then the existence of a directed path from an element fault to a PO is a sufficient condition for that fault to be *detectable* at that PO

VICTOR (Vlsi Identifier of Controllability, Testability, Observability and Redundancy) [Ratiu 82] is an example of a 4 pass linear algorithm for combinational logic circuits which endeavours to determine an UUT's testability, identify redundancy, and attempt to generate test vectors. VICTOR flattens (levels) the circuit, calculates the zero and one controllabilities, obtains the observability values and then indicates redundancies while performing test generation. However McCluskey notes that VICTOR [McCluskey 86] tends to be a pessimistic procedure,

identifying many nodes as being *potentially redundant* even if this is not the case.

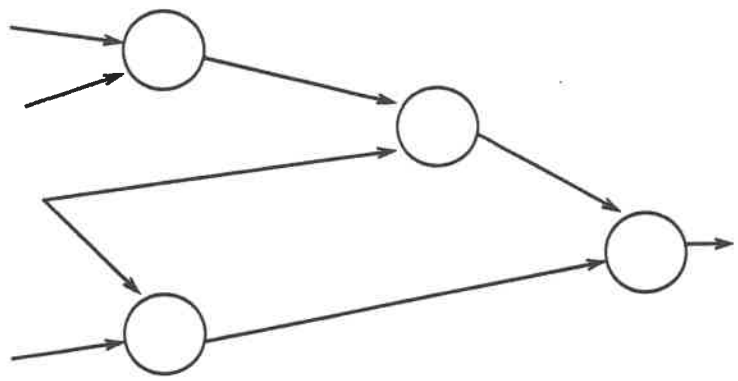


Figure 2.1.2a: Graph with reconvergent fanout.

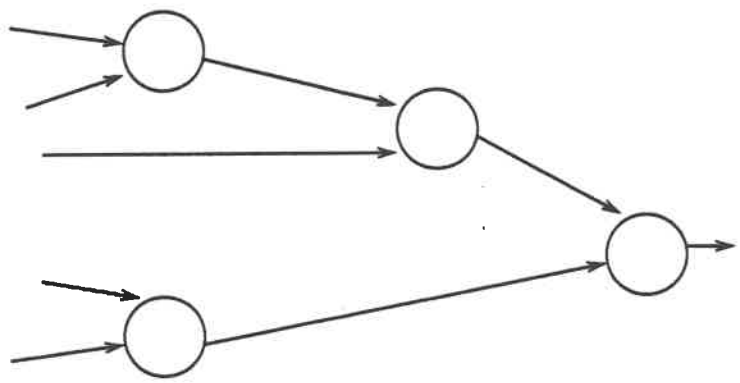


Figure 2.1.2b: Graph without reconvergent fanout.

2.2 Testability Measures (TMs).

Several researchers have tried to identify the presence of faults which will be *hard to test*, (HFs), by establishing *testability measures* (TMs). The testability of a design is evaluated using these TMs prior to generating the set of test vectors used to detect a faulty circuit. Efforts have focused on the creation of linear algorithms

based upon heuristics in order to quantify the relative difficulty in generating the subsequent testing stimuli. These TMs are derived, in general, from metrics designed to quantify the *controllability* and the *observability* of the individual *nets* or *lines* within the UUT. The *controllability* is the ability to set a given *net l* to a known logic value $v \in \{0, 1\}$ from values placed on the UUT's primary inputs (PIs). *Observability*, the counterpart of controllability, is the ability to observe the logic value on *net l* (typically with an error value impressed on it and not a logic 0 or logic 1) at a given UUT's primary output (PO). Observability is also a function of the nets' controllabilities as is shown below.

Poor controllability or observability hinders the ability to generate a test for a faulty circuit element. Lack of control can cause one or more of the following problems to occur:

- propagation of the error, caused by the faulty element, may be blocked due to a *blocking condition* on a logic gate: such as a logic 0 on one or more of an AND gate's inputs as shown in Figure 2.2.1;
- line justification may not be possible. Figure 2.2.2 shows a net with the current value of 'X' (don't care) and which requires setting to logic 1. In this example, a zero is more easily generated than a one, since only 1 of the 2^4 input combinations provides this logic 1;
- the actual sensitization of the error, local to the faulty logic element, may be difficult. The inability to generate an environment whereby the error condition is clearly evident may preclude the possibility of detecting the faulty circuitry.

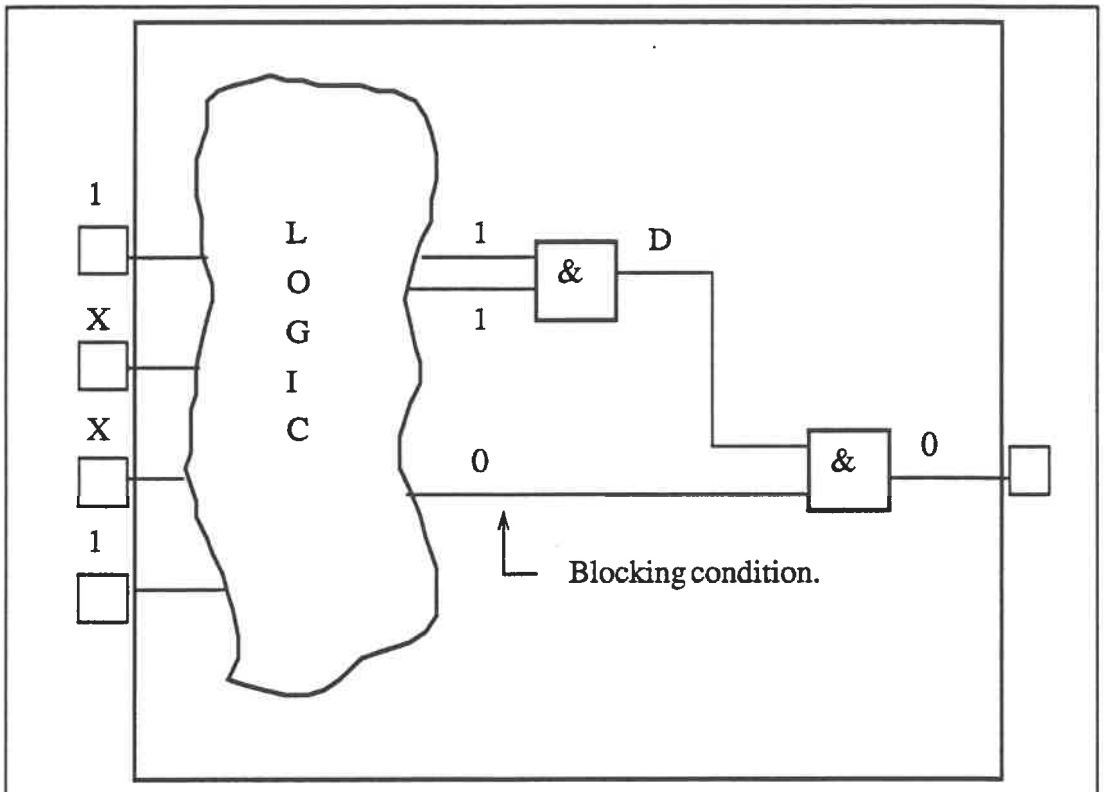


Figure 2.2.1: Blocking condition example.

Lack of observability can be related to the inability to control a net which would allow the error signal to be propagated to an UUT PO. The terms, *propagation, sensitization* and *justification*, will be explained in more detail as automatic test pattern generation is covered in Chapter 3.

Two examples of widely discussed TMs are presented: (1) SCOAP - Sandia Controllability Observability Analysis Program - which uses heuristics related to a circuit element's relative depth within the UUT; and (2) COP - Controllability Observability Program - that applies signal probability under the assumption of *independent fanout branches* for reconvergent fanout conditions. The overall advantage of these TMs is that they are O(n), linear, where *n* is the number of logic

gates contained by the circuit being examined.

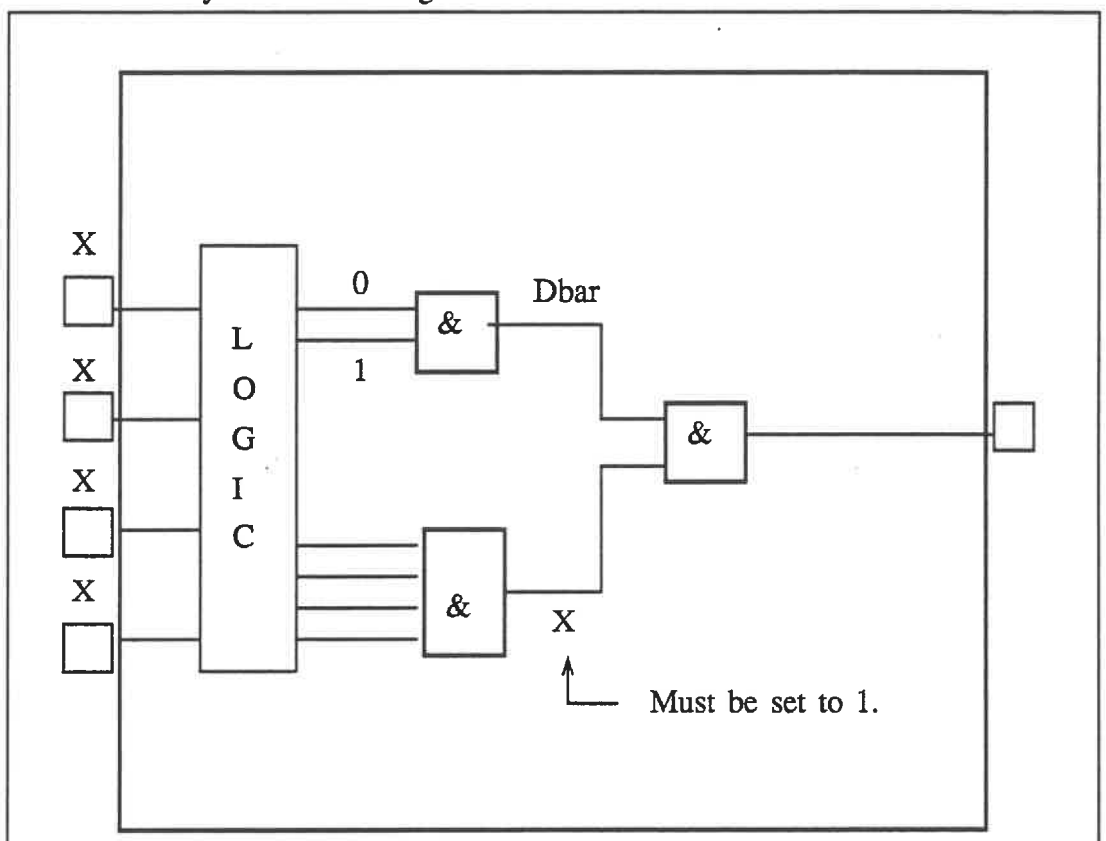


Figure 2.2.2: Line justification example.

2.2.1 SCOAP - Sandia Controllability Observability Analysis Program.

Goldstein suggested that the relative depth of a logic gate within the UUT would provide a measure of the ease in testing a fault on a net associated with this gate. SCOAP [Goldstein 79] [Goldstein 80] was the computer implementation of the algorithm which calculates the 6 TMs. These 6 metrics are used to gauge the testability of the combinational and sequential circuitry; only the 3 combinational TMs will be described. Essentially the controllability is decreased, indicated by an integer value which increases by an fixed quantity (tending towards infinity or poor control), as one proceeds from the UUT's primary inputs (PIs) heading towards the UUT's primary outputs (POs) through the various gates in its cone of influence.

SCOAP first determines the nets' *zero*, C_0(net *l*), and *one*, C_1(net *l*), controllabilities. This information is required for calculating the nets' observability on the second pass. The observability also decreases as the TM's magnitude increases

```
Initialize all PIs, POs and internal nets.
    C0 and C1 for PIs set to 1.
    C0 and C1 for POs and nets set to ∞.
    Obs for PIs and nets set to ∞.
    Obs for POs set to 0.
for all logic elements in UUT {
    Calculate C0.
    Calculate C1.
}
for all logic elements in UUT {
    Calculate Observability.
}
```

Figure 2.2.1.1: SCOAP pseudo code.

and is denoted by Obs(*l*); observabilities are independent of the value being observed. The SCOAP program's pseudo code is presented in Figure 2.2.1.1 makes use of the following relationships, only some are shown, and initial conditions in order to predict the testability. Some simple logic primitives, with their corresponding combinational controllability and observability equations, are shown in Figure 2.2.1.2. The same figure also provide the information for *fanout stems*, the net driving a fanout, and *fanout branches*. The initial conditions for the unit under test's primary inputs (PIs) and primary outputs (POs) are listed in Table 2.2.1.1. Agrawal has indicated in a past paper [Agrawal Mercer 82] the poor correlation between the predicted ease of testability and the reality.

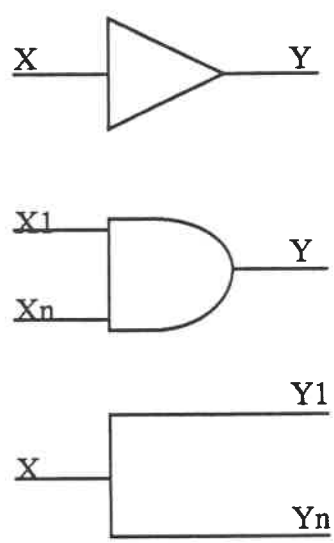
A simple circuit, the full adder of Figure 2.2.1.3, is used as an example to show the combinational controllabilities and observabilities for every net in the

circuit; these values are found in Table 2.2.1.2. This same circuit is reused in the next section, where a different set of testability measures is calculated. Note that the values within the accolades are for the fanout branches. The value chosen is the minimum of the set.

Parameters	*Primary Inputs*	*Primary Outputs*
C0	1	∞
C1	1	∞
Obs	∞	0

Table 2.2.1.1: SCOAP initial conditions.

SCOAP's sequential TMs are not discussed here as many other techniques have been proposed and since this thesis is primarily concerned with combinational logic HF identification. The TMs based upon sequential path length, as established from a graph representation seem to be a more favoured approach.



$C0(Y) = C1(X) + 1$
$C1(Y) = C0(X) + 1$
$Obs(X) = Obs(Y) + 1$

$C0(Y) = \min(C0(X_i)) + 1$
$C1(Y) = C1(X_i) + 1$
$Obs(X_i) = \Sigma_{j \neq i}\, C1(X_j) + 1$

$C0(Y_i) = C0(X) + 1$
$C1(Y_i) = C1(X) + 1$
$Obs(X) = \min(Obs(Y_i)) + 1$

Figure 2.2.1.2: Some elements and their SCOAP relations.

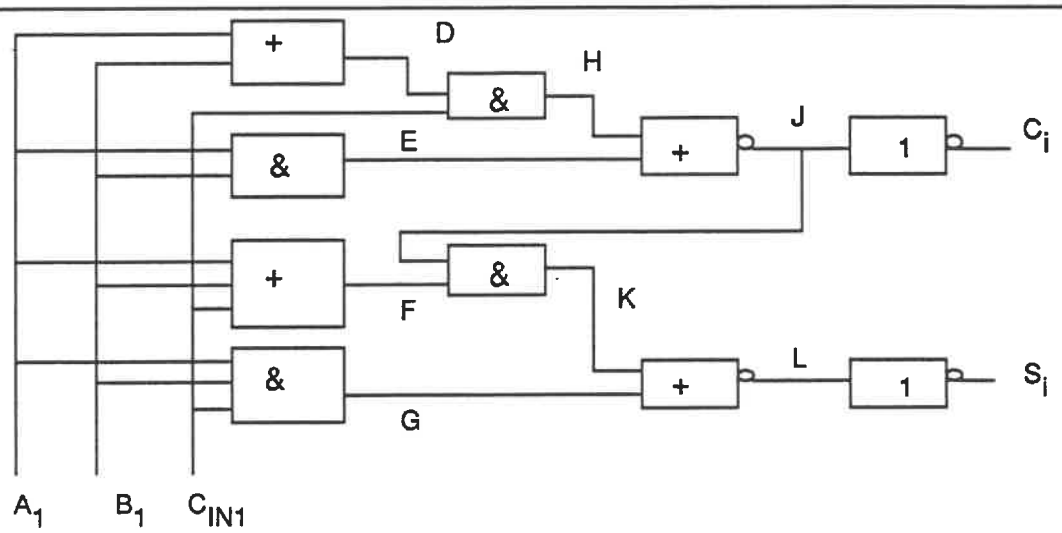


Figure 2.2.1.3: Full adder circuit.

2.2.2 COP - Controllability Observability Program.

Brglez suggested that a more correct method [Brglez 83] [Brglez 85] for determining a circuit's testability is to use signal probabilities in calculating the TMs. COP's controllability and observability metrics are restricted to only combinational circuitry resulting in a total of 3 measures. COP [Brglez 85] also requires two passes, highlighted in Figure 2.2.2.1, through a UUT to accomplish the testability analysis: the first pass is initiated from the UUT PIs and determines the controllabilities, C_0(net *l*) and C_1(net *l*) as the algorithm proceeds through the UUT towards the UUT POs; the second iteration determines the observability Obs(*l*) starting at the UUT POs and working towards the UUT PIs using the previously determined C_0(net *l*) and C_1(net *l*). The initial conditions are stated in Table 2.2.2.1 and some formulae are shown in Figure 2.2.2.2. The COP TMs, for the full adder circuit of Chapter 2.2.1, have been calculated and are found in Table 2.2.2.2.

NET NAME	C0	C1	Obs
A1	1	1	6 {8, 6, 13, 10}
B1	1	1	6 {8, 6, 13, 10}
CIN1	1	1	7 {7, 13, 10}
D	3	2	6
E	2	3	4
F	4	2	10
G	2	4	7
H	2	4	4
J	4	5	1
K	5	8	4
L	6	8	1
CARRY	6	5	0
SUM	9	7	0

Table 2.1.1.2: SCOAP values for the full adder circuit.

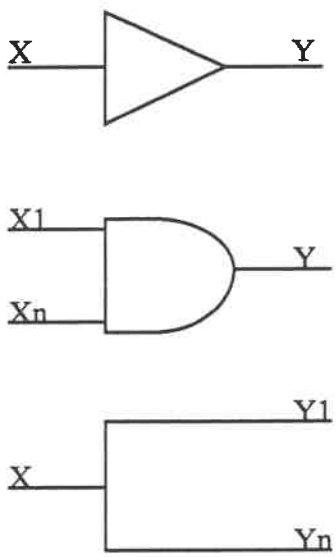
Parameters	*Primary Inputs*	*Primary Outputs*
C0	0.5	∞
C1	0.5	∞
Obs	∞	1

Table 2.2.2.1: COP initial conditions.

Other TMs have been suggested, but Huissman has indicated in a recent paper that there is still a great deal of difficulty in predicting a circuit's testability [Huissman 88]. He also notes that despite the lack of a definitive TM, and their lack of success, that work should continue in this area.

```
Initialize all PIs and POs.
    C0 and C1 for PIs set to 0.5.
    C0 and C1 for POs and nets set to ∞.
    Obs for PIs and nets set to ∞.
    Obs for POs set to 1.
for all logic elements in UUT {
    Calculate C0.
    Calculate C1.
}
for all logic elements in UUT {
    Calculate  Observability.
}
```

Figure 2.2.2.1: COP pseudo code.



$C0(Y) = C1(X)$
$C1(Y) = C0(X)$
$Obs(X) = Obs(Y)$

$C0(Y) = 1 - C1(Y)$
$C1(Y) = \prod C1(X_i) + 1$
$Obs(X_i) = \prod_{j \neq i} C1(X_j) \times Obs(Y)$

$C0(Y_i) = C0(X)$
$C1(Y_i) = C1(X)$
$Obs(X) = \min(Obs(Y_i))$

Figure 2.2.2.2: Some elements and their COP relations.

Despite the inherent limitations of these TMs caused by the use of heuristics, their use in guiding automatic test generators has become commonplace. These guidance heuristics have not been limited to the COP and SCOAP TMs.

2.3 Using of ATPGs to detect HFs.

Automatic Test Pattern Generation (ATPG) algorithms and procedures have been created to generate test vectors for a given fault set for a given circuit requiring testing. Algorithms, by their very nature, guarantee a solution (providing that there is at least one solution possible) if they are allowed to work until their natural termination point. However brute force methods such as the *D-algorithm* [Roth 67] may have a tendency to be highly inefficient for classes of logic circuits - and degrade to a time consuming exhaustive search of all the circuit's vector space. The assigning of decisions in a totally random manner lead to the development of improved techniques that prune the decision space: *PODEM* [Goel 81]. FAN [Fujiwara 83] used topological heuristics to reduce the decision space's size. PODEM and FAN

will become exhaustive searches in the limit should their pruning heuristics be useless, or of poor quality, for a particular circuit. Thus, an algorithmic ATPG can be used to decisively indicate an HF's presence.

As the ATPG works towards constructing a solution, it may encounter situations where a decision (arbitrary) is required; an incorrect choice is a costly effort as indicated by Fujiwara in his work on developing FAN [Fujiwara 83] [Fujiwara 85]. Reversing a previous decision - a *backtrack* operation on the decision space - requires that the ATPG perform much extra work. As such, many ATPGs will declare a fault as hard to test when the number of ''backtracks'', generated in the search for a solution, exceeds an arbitrary limit - such as 10. The fault may be incorrectly labelled since a generic heuristic has been arbitrarily imposed. This is, none the less, one additional method (above an exhaustive search) by which an ATPG can identify HFs - the metric of *cost of effort*. While the heuristic based result is not always precise, the exhaustive search is and one can safely assume that the cause is redundancy. A limit of 10 backtracks per test generation exercise for a selected fault appears to be a useful figure of merit. Researchers, such as Fujiwara, found that increasing this limit by 2 orders of magnitude does not significantly increase the quantity of detected faults. It would seem that the heuristics may be causing the ATPG failure. [Agrawal Seth 89] suggest that test generation is made difficult by factors other than the quantity of gates, the quantity of memory elements (sequential logic elements) and the circuit's sequential depth. Their list is augmented by Ivanov's suggestion that the heuristics [Ivanov 85] also compound the automatic test generator's quandary in finding a solution:

- poor initializability.
- poor controllability and observability for memory elements.
- structural dependencies (reconvergent fanout for example).
- cycles within the circuit.
- the guidance heuristics.

NET NAME	C0	C1	Obs
A1	0.500	0.500	0.312
B1	0.500	0.500	0.312
CIN1	0.500	0.500	0.562
D	0.250	0.750	0.375
E	0.750	0.250	0.625
F	0.125	0.875	0.410
G	0.875	0.125	0.560
H	0.625	0.375	0.750
J	0.531	0.469	1.000
K	0.590	0.410	0.875
L	0.484	0.516	1.000
CARRY	0.469	0.531	1.000
SUM	0.516	0.484	1.000

Table 2.2.2.2: COP values for the full adder circuit.

2.4 Previous use of cost in Testability Analysis.

2.4.1 Breuer's Sensitivity Functions.

Breuer used the concept of cost [Breuer 83] in extending the testability measure concepts. His specific goal was to identify points within the UUT for the optimal and automatic insertion of test points for improved observability and gates for increased controllability (addition of AND gates for improving the zero controllability and OR gates for facilitating the one controllability). He identified the cost of controlling nets and observing nets by heuristics that had served as the basis for SCOAP. Breuer labels the zero controllability as $¢_0(l)$, the one controllability as $¢_1(l)$, and the observability as $¢_2(l)$, for a given circuit net l. The UUT's costs are totalled by first summing each individual cost for L - the set of nets contained by the UUT - providing $\$_0$, $\$_1$, and $\$_2$ as shown in equations 2.4.1.1 - 2.4.1.3. The total testability cost, $\$'$, is the result of the weighted sum of these addends.

$$\$_0 = \sum_{l \in L} ¢_0(l) \qquad 2.4.1.1$$

$$\$_1 = \sum_{l \in L} ¢_1(l) \qquad 2.4.1.2$$

$$\$_2 = \sum_{l \in L} ¢_2(l) \qquad 2.4.1.3$$

$$\$' = \sum_{i}^{2} k_i \, \$_i \qquad 2.4.1.4$$

Breuer then specified a *sensitivity function* used to calculate the reduction in costs (such as the improved zero controllability of a net l obtained by inserting an AND gate) generated by adding a test point at an arbitrary circuit net. Equation

2.4.1.5 specifies the sensitivity for a single modification to a net *l*. S_j is a binary variable used to indicate that the test point is activated ($S_j = 1$) or that normal circuit functionality occurs ($S_j = 0$). Breuer [Breuer 83] [Chen Breuer 85] implemented a program by which constraints could be applied - the maximum cost of modifications; upper bounds on individual nets within the UUT - to the UUT while automatically applying DFT.

$$\frac{\partial \$_i}{\partial \$_j(l)} = \frac{|\ \$_i(S_j(l) = 1) - \$_i(S_j(l) = 0)\ |}{1 - 0} \qquad 2.4.1.5$$

2.4.2 Random Test Cost Functions.

Lisanke *et al.* also use a cost function - a testability cost function - while generating tests from randomly created patterns [Lisanke *et al.* 86] [Lisanke *et al.* 87]. Their program, *ESPRIT*, consists of the five modules indicated in the pseudo code shown in Figure 2.4.2.1.

```
do{
   Compute Input Signal Probabilities.      /* 1 */
   do{
      Generate Vectors.      /* 2 */
      Fault Simulation.      /* 3 */
      Update Fault List.     /* 4 */
      Evaluate Fault Coverage Slope.   /* 5 */
   } until Fault Coverage Slope < User's Value.
   Evaluate Fault Coverage Slope.
} until Fault Coverage Slope < User's Value.
```

Figure 2.4.2.1: ESPRIT pseudo code.

Costs are used for 2 of *ESPRIT*'s "functions": (1) - the computation of input signal probabilities and (2) - by the decision making process for continuing to attempt to generate tests. The former is the most important part of *ESPRIT* and is based upon COP [Brglez 84]. Remembering that the probability of detecting a fault on a net *l* is:

$$Pd_{l/0} = C_1(l) * Obs(l) \qquad 2.4.2.1$$

$$Pd_{l/1} = (1 - C_1(l)) * Obs(l) \qquad 2.4.2.2$$

One can then determine the *cumulative detection probability* CPd_j, by random patterns, given n independent trials, with the expression:

$$CPd_j(l) = 1 - (1 - Pd_j)^n \qquad 2.4.2.3$$

Lisanke *et al.* defined a *fault coverage estimate* (FCE) as a means to predict random test pattern generation costs for an UUT about which one knows the net list, fault set and input signal probabilities. The estimated fault coverage curve, a function of the number of independent trials is shown in Figure 2.4.2.2 and is the *average* of the cumulative detection probabilities over the fault set F as defined by:

$$FCE(n) = \frac{1}{F} \sum_{j=1}^{F} CPd_j(l) \qquad 2.4.2.4$$

That the estimated fault coverage approaches unity with an increasing number of trials is intuitive - as n approaches 2^m (m primary inputs on the UUT) the testing becomes exhaustive.

The area above the curve described by FCE is proportional to the amount of effort required by the fault simulation phase, Since ESPRIT's run time costs (computer processing unit's - CPU - time or cost) is dominated by the fault

simulation phase, the authors created a cost function $ that represents this area $. This area is defined by the equation:

$$\$ = \sum_{n=0}^{\infty} [1 - FCE(n)] \qquad 2.4.10$$

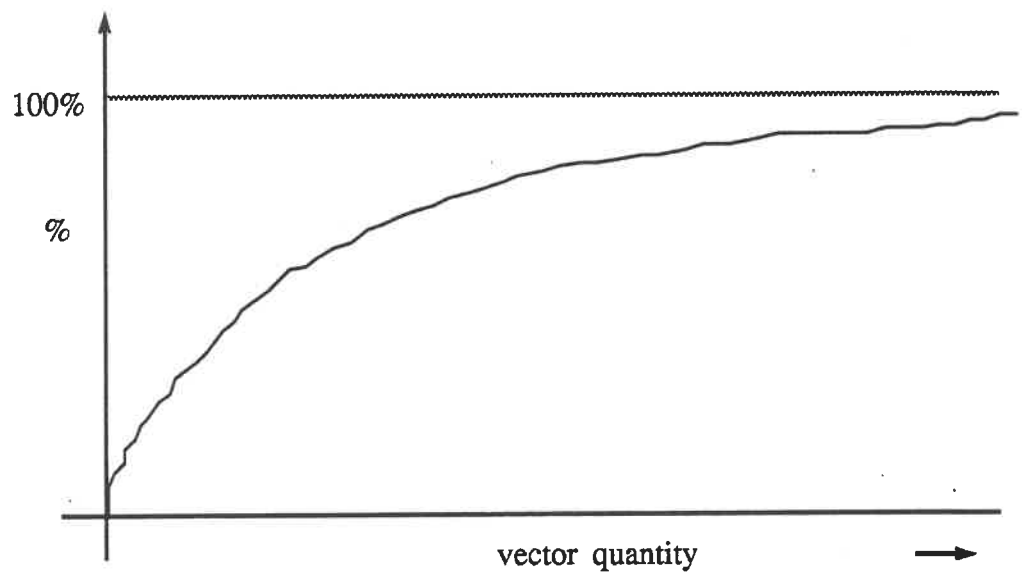


Figure 2.4.2.2: Fault Coverage function.

The authors minimize the cost function $ in order to produce the optimal input signal probabilities used during the vector generation phase. This is achieved by using a gradient optimization technique that provides information on how $ changes due to input signal probability modifications.

2.4.3 Sequential Test Cost Functions.

Agrawal *et al.*, in creating a new ATPG [Agrawal *et al.* 89], defined three (3) cost functions used in developing a generalized directed search methodology. (1) An *initialization cost function* is used during the circuit initialization phase of their ATPG. (2) A *concurrent test generation cost function* is employed for simultaneously

testing several faults, and (3) a *single fault test generation cost function* is used when testing a single fault while generating vectors. Each of these cost functions is applicable to one of the three independent ATPG phases.

The initialization phase is required to generate an initialization vector sequence where the number of unknown signals on flip flop logic elements increases the *initialization cost.* The user may specify a non-zero initialization target value instead of the default target of zero cost.

A distance metric, the shortest distance from *any* fault effect of that fault to *any* primary output, is used in selecting whether a vector candidate is to be accepted after its trial. Agrawal *et al.* contend that the smaller the cost, the closer the fault is to being detected - thus a detected fault has zero cost.

The third cost function - *single fault test generation cost function* - is similar to the concepts proposed by the SCOAP TM. This dynamic cost reflects the *minimum* number of primary inputs that must be changed and the minimum number of additional vectors to control a node's value. This cost must describe both the effort to sensitize the fault and to propagate the fault to a primary output. A weighting penalty may be assigned to the dynamic sequential controllability (DSC).

2.4.4 Summary of Cost Functions.

The above three methods are indicative of the segregated approach to the use of cost in the testability arena. Breuer used his cost metrics to indicate how design for testability could be automatically added to the UUT. However, Breuer's metrics were based upon SCOAP's predecessor and Agrawal [Agrawal Mercer 82] has previously demonstrated the limitations of SCOAP TMs. [Agrawal *et al.* 89] also rely heavily on heuristic cost measures although attempts were made to vertically integrate these measures into an ATPG system. Lisanke *et al.* attempted to unify the cost of test pattern generation, albeit randomly created test patterns, with TMs.

An improved technique would appear to be a properly formulated amalgamation of these seemingly disparate methods; testability measures,

heuristics, costs, automatic test pattern generation and user supplied constraints. Cost is a primordial consideration given that one cannot:

- increase the circuit size without reasonable justification;
- degrade system performance;
- monopolize computer capacity for ATPG;
- cause excessive test application times due to excessively long test patterns;
- allow poor quality product, determined by lower fault coverage values, to reach clients.

Many of these, if not all, cost related parameters can be used to define a more precise definition of device testability and may be applicable on a varying product by product basis. This shall be the thrust of this thesis - the identification of HFs based upon budgetary (cost) constraints.

3.0 Review of Automatic Test Pattern Generation (ATPG).

ATPG principles are reviewed in depth in this section, providing the foundation for the HF identification algorithm *HUB*. The traditional fault model, the *single stuck at fault*, and the use of the five valued logic - at the heart of the D calculus used for the D algorithm - are also presented. This section also reviews the differences between the single path and multiple path sensitization methodologies. The definitions of basic terms are interspersed throughout the section. An ATPG's use of heuristics is explained as are the trade-offs and additional features encountered during the creation of an automatic test pattern generator.

3.1 Fault modelling.

Dr. Tom Williams noted that the concept of testing structure instead of function, as proposed by R. Eldred, was responsible for the literal explosion of research into fault models and the accompanying automatic test pattern generation tools [Williams 89a]. Eldred described how structure could be verified based upon tests for specific *faults* under a *fault model* which accurately describes the realistic set of physical defects that occur within the structure [Eldred 59]. An underlying theme of fault models is the 3 abstraction levels associated with the concepts of *defects, faults, and errors.*

Defects refer to the *physical aberration* or anomaly which occurs during the manufacturing process. For integrated circuits, these may correspond to contaminants, metallization problems (voids or bridging), contact problems, etc.. These defects can have different defect densities and are often a function of the particular process [Galiay 80] . The defects may be modelled as having an influence over a relatively large area - known as *global* or *clustered defects* - or as very localized defects - *point defects* [Shen 85]. A metal bridging defect between the positive power supply and an inverter's output is shown at a layout representation level in Figure 3.1.1.a. This defect's effect could be represented at the transistor abstraction level by a *fault* such as the well known *single-stuck-at-n*, $n \in \{0, 1\}$ as

shown in Figure 3.1.1.b.

In turn, the fault's presence causes the introduction of an *error* at a functional abstract level. As an example, if this inverter was one of an instructions ROM's outputs and was fed to the input of a signature register [Bardell *et al.* 86], one would obtain a *burst error* [Stallings 88].

There are several advantages in using a fault model. The following are some of those enumerated by [Miczo 86]:

- Create tests specifically for those faults most likely to occur;
- Compute a *test quality metric* by determining how many potential faults are caught by the test stimulus;
- Debug yield problems by relating defects to specific test patterns.

Considerable work has been done recently, using a technique known as Inductive Fault Analysis (IFA), in relating the fabrication process knowledge to the circuit layout in order to determine the fault types most likely to occur [Shen 85] [Ferguson 88].

3.1.1 Single stuck at fault model and 5 value logic.

A typical defect in the *MOSFET* (Metal Oxide Surface Field Effect Transistor) technology occurs in the metallization [Galiay 80] e.g.: the bridging of two adjacent metal runs. This might cause a transistors output to be shorted to the power supply. Such a defect is typically represented by a *fault* such as the well known *single stuck at n*, *s-a-n* where $n \in \{0, 1\}$ and it is assumed that only one such fault exists at any given time. Figure 3.1.1a. represents an inverter, in *CMOS* (Complementary MOS), which has its output shorted to the positive power supply. The corresponding fault at transistor and gate abstraction levels, a *s-a-1*, are indicated in Figure 3.1.1b and Figure 3.1.1.1.

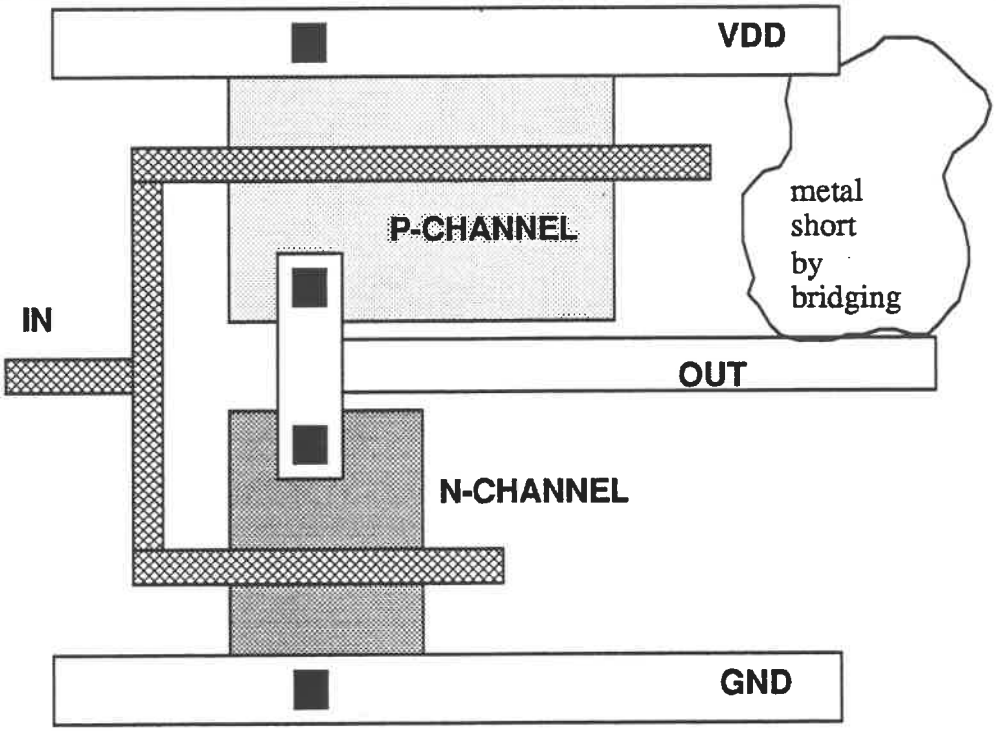


Figure 3.1.1a: CMOS inverter with output shorted to VDD.

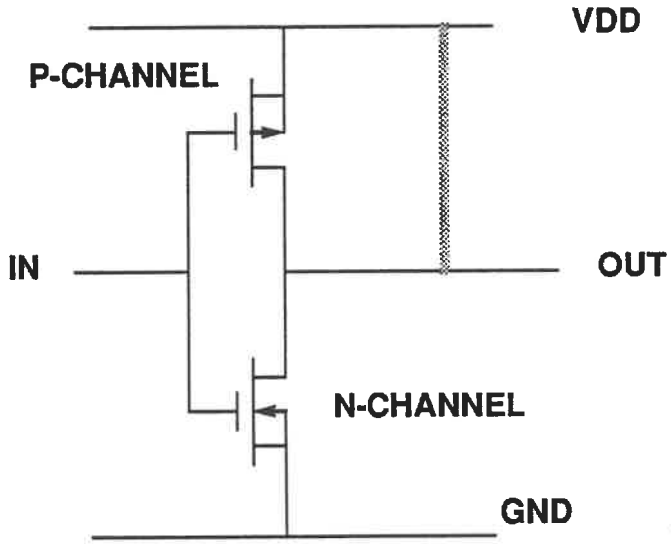


Figure 3.1.1b: CMOS inverter with output shorted to VDD.

One wishes to detect the presence of improper circuit fabrication in an efficient manner. The tendency has been to use a fault model - a wide variety exists - especially the single stuck at n fault model which was created around the late 1950s [Eldred 59] and whose use, correctly or otherwise, has continued into this age of *CMOS* and *BiCMOS* (Bipolar CMOS mixed technology process). It has been suggested repeatedly that the stuck at model is insufficient for CMOS technology and lead to the development of the *stuck open* [Wadsack 78]. However, simulators for the single stuck at fault model are seemingly the dominant factor. Automatic test pattern generation is typically constructed for *s-at-n* faults. It is important to note that stuck open faults appear to be correctable through proper design approaches [Soden 89], are apparently detectable when a high single stuck at fault coverage exists simultaneously with nodes that change states frequently - *high toggle activity*, and that their occurrence is not as high as the literature had previously suggested [Shen 85].

Indeed, it has been suggested that certain structures such as PLAs may not be adequately modelled by the *s-a-n* regardless of the process technology.

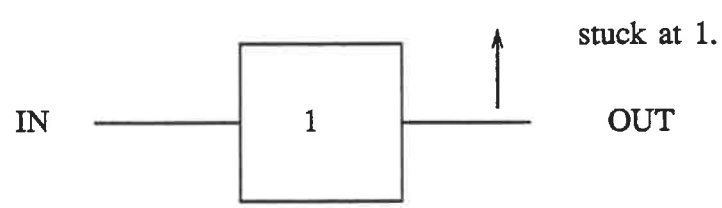


Figure 3.1.1.1: CMOS inverter with output stuck at 1.

The presence of a stuck at fault is commonly indicated by the use of a 5 valued logic system where any net *l* within the circuit, can have the value $v \in \{0, 1, X, D, Dbar\}$. The D signifies the presence of a *D*ifference between a correctly fabricated and an improperly manufactured circuit. The intersections producing D and Dbar are shown in Table 3.1.1.1. This table shows 2 important concepts: (1) - the presence of a fault can only be indicated by the detection of a difference; (2) - that the difference

is only specified for the logic values 0 and 1.

FAULTY CIRCUIT \ GOOD CIRCUIT	0	1
0	0	D
1	$\overline{D}$	1

Table 3.1.1.1: Rules for Difference intersection.

To successfully create a test for a given fault, it is essential to cause the difference D or Dbar, to be visible on at least one of the unit under test's primary outputs. Thus the difference must exist at two levels: (1) *locally* - that is for the sub element within the UUT which is the target for a fault and (2) *globally* - that is visible external to the UUT. Failure to comply with either requirement will cause the fault to be untestable.

Given a circuit to be tested, UUT, a set of faults can be created - typically this is at a gate level representation. The *fault list* or *fault dictionary F(UUT)* is generally unranked [Shen 85] meaning that any relative importance for the different faults is ignored. This fault dictionary may not be minimally sized since faults may be *collapsed* (reduced) on local and global levels. Generally *n-input* logic primitives such as AND and OR gates have their fault quantity reduced from $2n + 2$ to $n + 2$ faults on a local level [Miczo 86]. Further fault collapsing can be obtained by using *fault dominance* [McCluskey 86] and *fault equivalence* [Miczo 86] relationships. Indeed,

at the local level, many *n-input* combinational logic functions can further reduce their faults to $n + 1$.

3.2 Single versus Multiple path sensitization.

For a given circuit, the difference value may arrive at a *fanout stem* (a net which controls more than one destination logic element) as effort is expended, to cause the difference to appear on a primary output (PO). This scenario is depicted in Figure 3.2.1. One could chose to provide a path from this fanout stem to an PO by selecting a unique path, known as *single path sensitization* (SPS), or by selecting several paths, known as *multiple path sensitization* (MPS). In the case of SPS, "all" that one need do is to pick the best path - minimum effort (easiest) or least costly - among the legitimate choices available. MPS typically attempts to drive the error towards the UUT's POs by all possible valid paths. Picking the winning route in SPS is usually done with the aid of heuristics - indeed the use of testability measures such as the COP observability measure. The original version of PODEM [Goel 81] used a distance metric similar to SCOAP while Ivanov used COP in a later version [Ivanov 85]. A gate with an "X" on its output and having an error signal on one of its inputs, is known as a *D Frontier*; this D Frontier is the starting point for the path sensitization phase.

3.3 The Four ATPG phases of Deterministic Gate Level Test Generation.

Given an arbitrary unit under test for a given fault, such as shown in Figure 3.3.1, the objective is to generate a test assuming the presence of a fault - for this example and the remainder of this thesis unless expressly stated to the contrary, all test generation shall assume the single stuck at fault model - which will detect its presence in an improperly fabricated circuit. The element, an 2 input AND logic gate, is selected as the target for a fault represented as D. Prior to the test all internal nets are assumed to be Xs as are the UUT's PIs and POs.

Any ATPG can be reduced to a set of four basic operations when generating a test for a given fault. The *sensitization* (or error activation) of a fault for the element

within the UUT is required to render the fault visible - by the presence of a difference. Although for simple logic primitives, such as ANDs, ORs etc., this will only be done once per trial due to the small fault sensitization set, the sensitization phase can conceptually occur repeatedly if one cannot subsequently create an environment within the UUT by which a valid test is created. Once the difference has been created, the three other phases are engaged: the *error propagation* using either single or multiple path sensitization; *justification* of the nets' values which are set during the various phases and the evaluation of net values or *implication* phase. These phases may occur in a variety of sequences - this is an implementation issue.

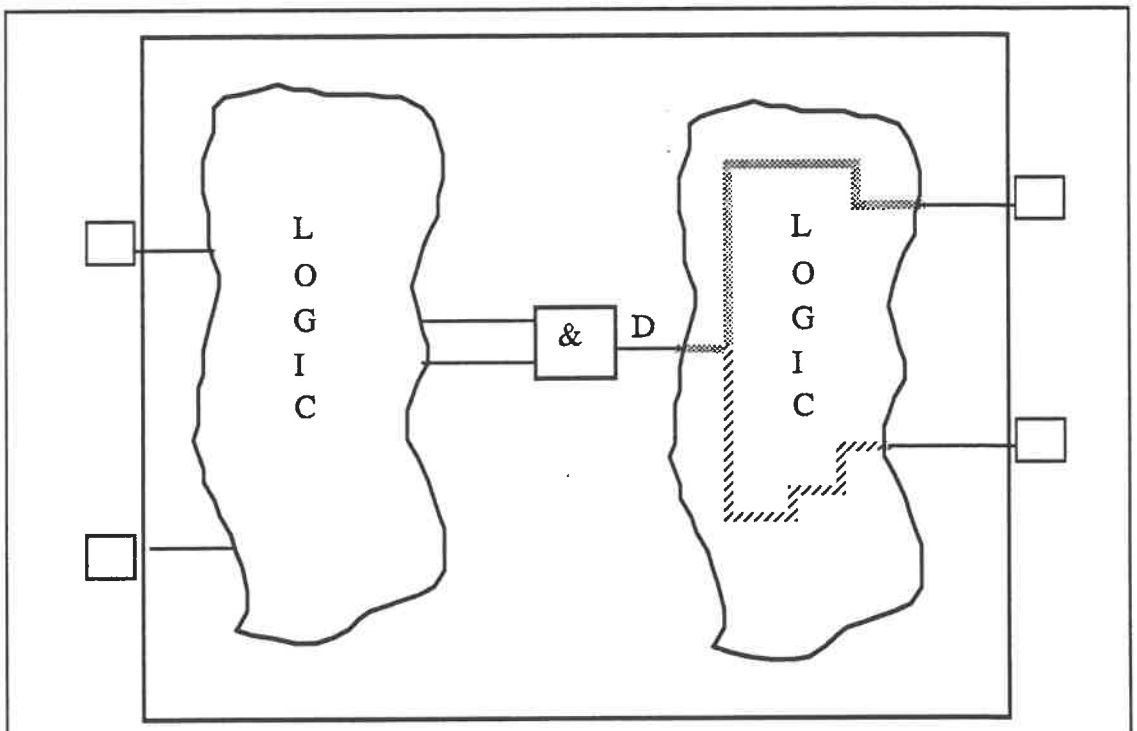


Figure 3.2.1: Path sensitization example.

To sensitize the *s-a-0* error on the AND gate's output, Figure 3.3.1, it is first necessary to generate the difference D requiring that the fault free AND gate produce a logic 1 - refer to Table 3.1.1.1. This dictates that all the AND gate's inputs must be set to logic 1 for both the fault free and the faulted circuit. This step is referred to as the *error sensitization* or *activation phase*.

Once a net's value has been uniquely assigned ($l \in \{0, 1\}$), an *implication phase* may be invoked. Implication is simply an evaluation (*simulation*) of the net's effect on other logic elements within the circuit and within its cone of influence. Since no fanout of the AND gate's inputs occur in this example, no implication is required due to the error sensitization. This simulation may occur whenever a list of nets, whose values have changed, exists. This phase can discover the presence of *conflicts* - a node that must take inconsistent values simultaneously: 0 and 1; D and 1 or Dbar and 0.

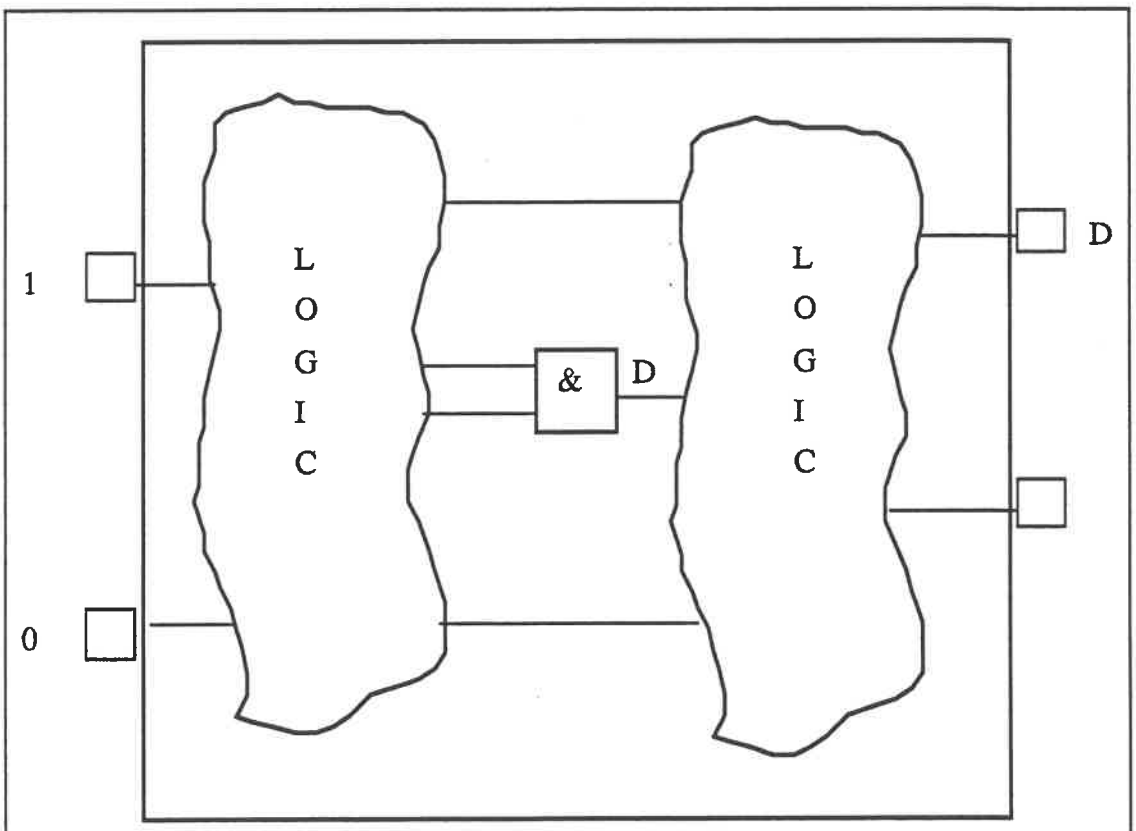


Figure 3.3.1: ATPG example.

It is necessary to propagate the error, using an *error propagation phase*, until the error appears on at least one of the UUT's POs. Either single or multiple path sensitization may be used, and possibly a mixture of both. Figure 3.3.2 shows how the error propagation might occur.

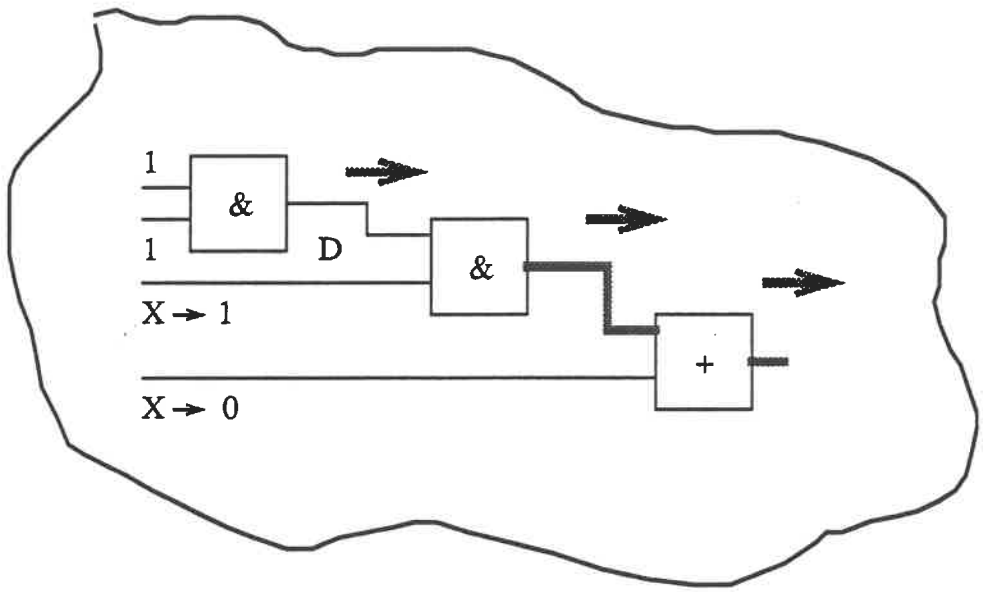


Figure 3.3.2: Error propagation example.

It is also required to justify the assignment of net values. Net values are assigned when the error is sensitized and as the error is propagated at each logic primitive en route to an UUT PO. Figure 3.3.3 indicates how a typical justification phase might occur.

Thus, in general, one could represent the ATPG's actions by the pseudo code of Figure 3.3.4. The simplified code does not consider how the search space is created or how reversing previous decisions is handled. Also, it is not necessarily true that this order is followed exactly as there may be a variety of implementations. It is possible to add a fault simulation phase to identify other faults detectable by the test vector (using the deterministically generated stimulus). Fault simulation is one method that allows a more compact test set by targeting easier faults and without reverting to the use of vector manipulation routines after the completion of the test generation phase. Previous authors have indicated that a good percentage of the

UUT's fault set may be easily detectable by random vector generation - this is suggested by the rapid rate of fault detection early in the ATPG; it has been suggested that this is roughly in the 65% to 85% fault coverage region [Agrawal Seth 88]. Even in the case of sequential test pattern generation, one can find these central phases [Marlett 88].

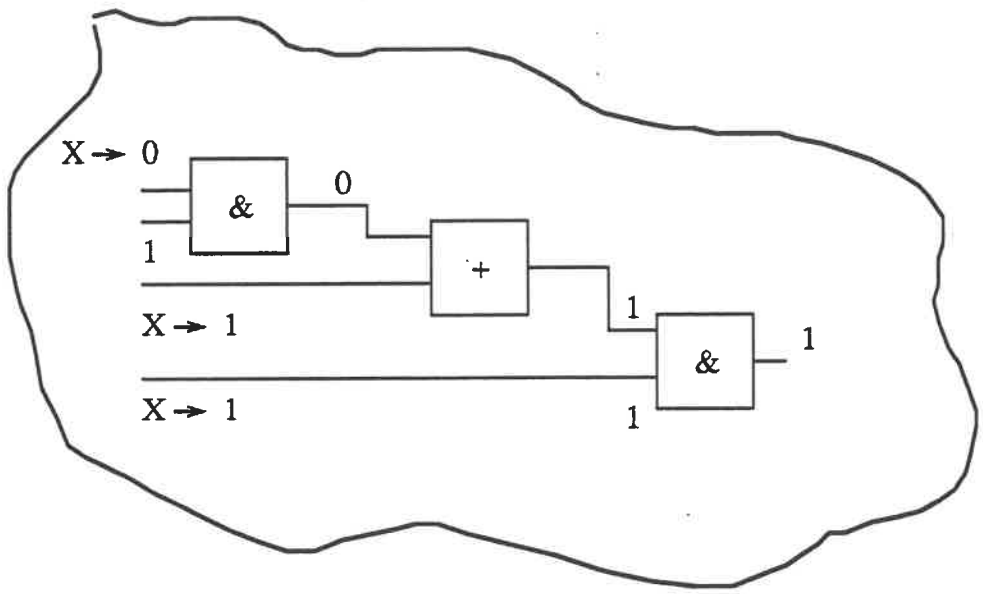


Figure 3.3.3: Net justification example.

The ATPG works backwards in the circuit graph to build the *decision space* or the *tree* that represents the search space for the fault in question by a process known as *backtracing*. While the circuit graph is immutable for the whole circuit during the test generation process, the decision space depends upon the target fault and the search algorithms (or procedure in the case of a non algorithmic program implementation). Should a decision be unjustifiable or create a conflict, the decision must be changed through a technique called *backtracking*. Although different methods might be used, driven by complex heuristics, the general principle is to change the last unchanged decision which will eliminate the pending conflict and which will not generate any new conflicts. Should no choices remain, an explicit or implicit exhaustive search of the decision space has occurred; this determines that the fault is

untestable due to redundancy which was previously seen to be caused by the presence of reconvergent fanout. Large backtracks usually trigger a limit and abort the search for the targeted fault.

```
while (Faults remain in UUT){
   Pick a fault.
   Sensitize the fault.
   while (No test && Choices remain){
      Propagate fault to an UUT PO.
      Justify all nets' assignments.
      Implication of nets' assignments.
   }
   Simulate for other faults detected by this test.
}
```

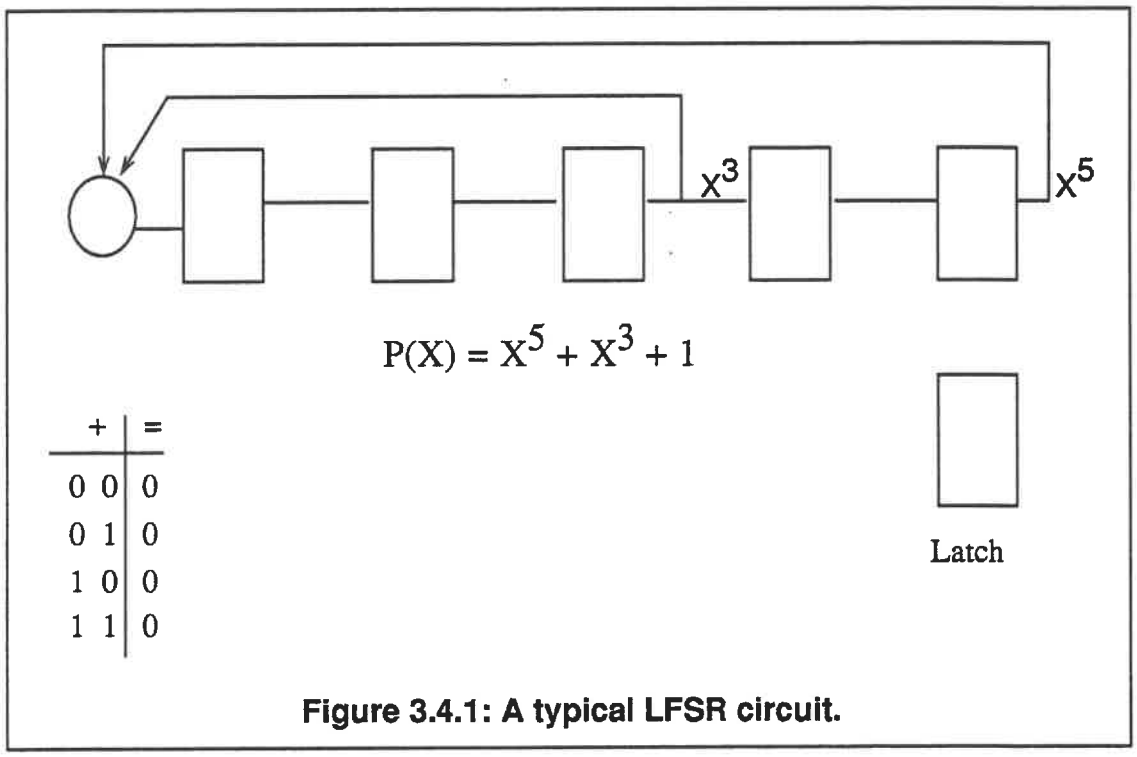
Figure 3.3.4: A typical ATPG algorithm.

The decision space's size, if one considers the UUT as a black box with n PIs, could consist of *n* nodes which are a 1:1 mapping of the PIs and their state. This is precisely the case for PODEM [Goel 81] while the D algorithm's size can potentially include a node for every circuit net. A tuple placed on a stack can be effectively used to represent the search space in a computer representation.

3.4 Random Test Pattern Generation and Hybrid Methods.

It has been suggested that the first portion of deterministic test generation, where each vector may detect many additional faults (hence the use of fault simulation), is due to the *random easiness* of these faults. Therefore these faults could be detected by using randomly generated test vectors. Many authors have

worked on such techniques [Bardell *et al.* 86] especially since much work in the *Built In Self Test* (BIST) area is centered on the use of pseudo random stimulus created by using LFSRs (*Linear Feedback Shift Registers*) such as that in Figure 3.4.1. Further work on varying the outputs weighting has occurred in order to allow nearly 100% fault detection.



+		=
0	0	0
0	1	0
1	0	0
1	1	0

Figure 3.4.1: A typical LFSR circuit.

An additional approach has been to use a two phase methodology: detect the random easy faults using randomly generated test vectors - performing fault simulation on these vectors; then switch to a deterministic automatic test pattern generator to detect the remaining faults. This method is depicted by the pseudo code in Figure 3.4.2. A recent article [Abramovici 89] suggests that this hybrid technique is of little or no use. Their research shows that random test vectors detect the same faults found in the second phase.

```
while (Faults remain in UUT){
   while (Random easy){
      Create a random test vector.
      Perform fault simulation.
   }
   while (Not random easy){
      Deterministic ATPG.
   }
}
```

Figure 3.4.2: Random Test Pattern Generation.

Another approach to test pattern generation was suggested by Chang *et al.* in a hierarchical test generator CHIEFS [Chang *et al.* 86]. This ATPG used a technology independent circuit representation called *binary decision diagrams* (BDDs) [Akers 78a] which also allowed test generation for circuits about which no detailed design knowledge was available. The tests were built around the ability of the BDDs to represent the logical function of the circuit, readily constructed from the circuit's truth table. These *experiments* were designed to elicit stimulus that tested for correct operation and to test that undesired functionality was not present. Although obtaining good test coverage using this new fault model, the fault coverage using the single stuck at fault model tended to suffer - albeit that from one implementation method to another for the same circuit more consistency was achieved than by predecessors [Abadir Reghbati 85] [Abadir Reghbati 86]. Chang noted that this traditional fault coverage improved with increased implementation knowledge. Note that a new simulator was required to support this circuit and fault model. Until recently, this has been one of the few attempts to generate tests where hierarchy has been exploited.

MODEM [Calhoun Brglez 89] is a recent example of a modification to an existing tool and methodology, that is PODEM, where an attempt to reduce the ATPG effort is constructed around the use of hierarchy. Since hierarchical representations will reduce the quantity *n*, the exponent in the computational complexity equation, effort will probably continue along these lines during the coming years. There has even been research into reuse of existing test stimulus for *macros* - a larger circuit constructed recursively from instances of logic gates and other macros - since one can amortize the test development effort for the macro over larger amounts of designs; [Somenzi *et al.* 85] is one such example.

3.5 Review of the FAN algorithm.

There have been three ATPGs whose names re-occur regularly in the test literature: the D-algorithm, PODEM, and FAN. Roth's D algorithm [Roth 67] is based upon his D calculus and provides a complete method for generating test vectors for digital circuits. Unfortunately his algorithm tends to be considered as highly inefficient for certain classes of circuits [Miczo 86] such as EDACs (Error Detecting And Correcting) where large number of modulo 2 logic primitives may be found. The D-algorithm, when faced with a decision, did not use any heuristics and very quickly became an explicit exhaustive search of the decision space due to the poor choices made.

PODEM [Goel 81] uses heuristics and a different approach for combinational logic circuits: a branch and bound algorithm. Goel's method increased the ATPG's performance by simultaneously reducing the required computer run time - due to an implicit enumeration methodology - and by reducing the amount of data retained by the ATPG. This reduction in memory requirement was achieved by reducing the number of nets involved in the ATPG decision process from potentially all internal circuit nets (as is the case for the D algorithm) to only those of the UUT's PIs. This meant that a decision tree built from the UUT's PIs was employed for the implicit enumeration as opposed to a cube containing, potentially, the vertices of every circuit net.

Fujiwara made the astute observation that PODEM's performance could be further improved by reducing the *backtrack* quantity (the re-evaluation of a previous decision due to a conflict arising in the future) caused by the fanout contained within the UUT [Fujiwara 83]. Specifically, Fujiwara attacked the quantity of backtracks generated during the ATPG process and the time needed to detect and process the backtracks. The literature suggests a performance improvement of 5 to 6 times over PODEM. A similar value with respect to PODEM's improved performance over the D algorithm may hold [Seth Agrawal 85]. It is important to note that one is referring to linear improvements applied to an NP complete problem.

The resulting algorithm, FAN [Fujiwara 83][Fujiwara 85], is shown in Figure 3.5.2. One can find the four principal ATPG phases described earlier in section 3.3. Three key types of net classes are identified in order to aid and regulate the various test generation stages, shown in Figure 3.5.1:

- BOUND LINEs: those nets reachable from some fanout stem.
- FREE LINE: a net which is not bound.
- HEAD LINE: a free line adjacent to a bound line.

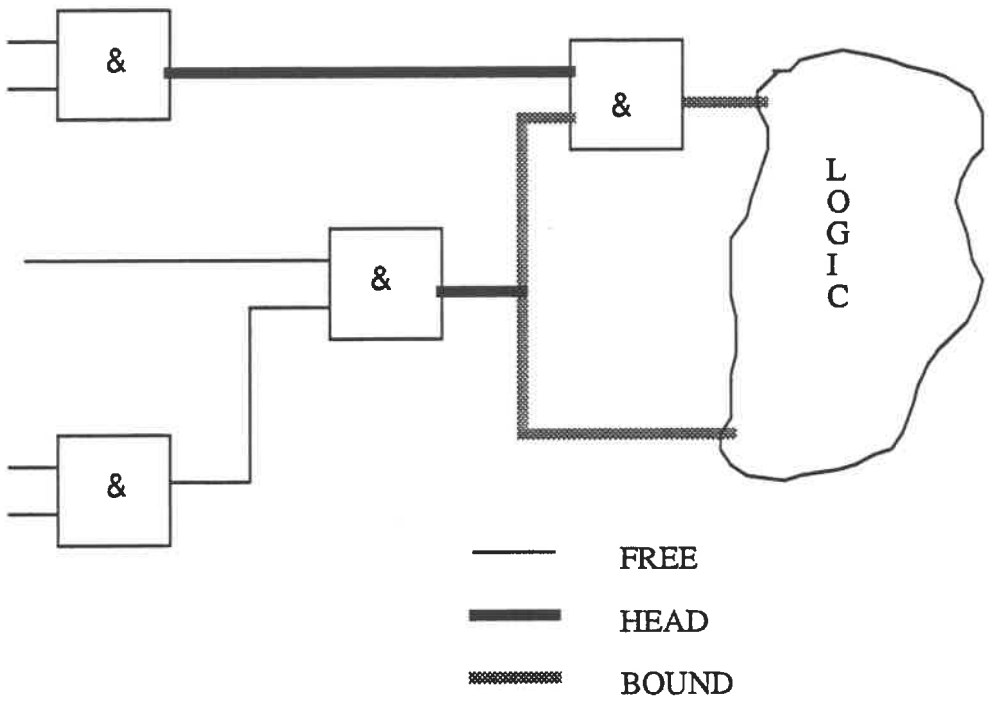


Figure 3.5.1: Types of nets in FAN.

The ATPG algorithm FAN [Fujiwara 83] [Fujiwara 85] is described in pseudo code in Figure 3.5.2. Note that the four principal ATPG phases are found.

FAN traces backwards in the UUT towards the PIs - *backtrace* - until it encounters a head line. Head lines and free lines are justified once all bound lines have been justified since, due to their fanout free nature, they are guaranteed to be justifiable. FAN constructs a *decision space* based upon head and free lines; this decision space is used to control *backtracks* if and when a previously made decision causes a conflict or a block propagation path to occur.

```
while (Faults remain in UUT){
    Pick a fault.
    Sensitize the fault.
    while (No test yet & Not exceeded backtracks){
        Implication.
        if (Error at UUT PO){
            if (No unjustified BOUND lines){
                Justify FREE and HEAD lines.
                Simulate for other faults detected by this test.
            } else BackTracE.
        }
        else if (Error not at UUT PO){
            if (D Frontier > 1 gate){
                Multiple  BackTracE.
            }
            else if (D Frontier = 1 gate){
                Unique Sensitization.
                BackTracE.
            }
            else  BacKtracK
        }
    }
}
```

Figure 3.5.2: FAN pseudo code.

3.6 Backtrack reduction methods and the importance of heuristics.

Backtracks are an important consideration in the ATPG process: they can cause a target fault to be classified as untestable or an HF by aborting the search for the targeted fault; they also are a cause of performance degradation due to increased processing time requirements. Backtrack reduction has been the subject of much research and was the principle reason that PODEM and FAN were created. Some authors, [Marlett 89] [Bell Taylor 88] have stated that it is important to make the correct choices and thereby avoid or reduce the backtracks. They have heuristically attacked the backtrack causes in order to increase the percentage of faults detected by the ATPG and thus reduce the number of aborted faults. Ivanov has suggested that the guidance heuristics employed by the automatic test pattern generator can effect the problem; the use of COP TMs in setting PIs in the case of PODEM is an example.

Another approach has been to target the search strategy since research on the test generation algorithms suggests that there is no perfect general search strategy. An example of a search strategy is in PODEM where the D Frontier closest to a PO is selected for error propagation purposes. Patel tried multiple search strategies based upon different testability measures [Patel Patel 86]. A recent article has proposed that the use of switching search strategies based upon the ordering of the ATPG phases while striving to maintain a low backtrack limit appears to be successful in exploiting the characteristics of different circuit's [Min Rogers 89]. This was tested on the ISCAS 85 benchmark circuits using a modified version of FAN.

It was decided early in this research that the multiple backtrace option of FAN would not be implemented. This decision would seem to be supported by the experience of other researchers: for example [Marlett 89] indicated that from his practical experience, little is gained from the extra work; Min further substantiates this by his research which shows that the multiple backtrace is not better than single backtrace - it is more CPU intensive though [Min Rogers 89].

3.7 Summary.

The basic concepts of deterministic automatic test pattern generation and the associated fundamental definitions have been presented. The principles of fault modelling and how they are typically represented, the single stuck at fault model, were reviewed to show how the four principle ATPG phases function. Additional test generation techniques were presented prior to summarizing Fujiwara's FAN ATPG algorithm which serves as the basis for the HUB hard fault identification tool. The importance of the backtrack phase and the need to reduce these backtracks were discussed.

4.0 Mixed Graph - Binary Decision Diagram (gBDD) Circuit Model.

It is necessary to describe the circuit at various stages of the design. A netlist is used to describe the components from which the circuit is constructed (AND, OR, INVERTER gates for example) and define net connectivity information. Such descriptions, while providing a convenient method to represent the circuit textually, are not necessarily the internal representation employed by the various design tools: electronic schematic capture, automatic test pattern generators (ATPG), and logic and fault simulators. It was imperative to provide such a representation to the HUB (Hard fault detection Using Budget constraints) system.

During the literature research on efficient ATPGs, one could determine how the test vector generation algorithms and procedures worked and might be implemented. There was, by comparison, no such definitive information on how the circuit related data would best be modelled: circuit structural or connectivity data, logic element functional modelling, fault modelling and fault dictionary representations were apparently left to the devices of each team performing the implementation.

To support the desired objectives for HUB, it was felt that a circuit description for this hard fault detection algorithm must provide certain key attributes if the proposed method was to be successful: (1) include structural circuit information; (2) allow functional intent and functional modelling to become an integral part of the model; (3) support the faulting of nodes within the circuit; and (4) permit the logic simulation phase of an ATPG. While one may conjecture that other similar work is based upon graph techniques, such speculation is left to specific works researching circuit modelling methodologies.

4.1 Introduction.

The UUT's representation during the ATPG process can have an effect on its efficiency. Too much detail, such as working at a transistor implementation level, while providing much implementation related information will use an excessive

amount of computer memory. This, when coupled by large decision spaces caused by the *backtrace algorithm*, can cause the computer to spend much time on overhead activities - swapping pages of memory or page faults. It is important therefore, to choose an abstraction level which provides sufficient information but only that required by the test generator. This knowledge should aid the decision making heuristics, and given the hierarchical design environments proliferating to help manage the design task, allow the introduction of hierarchy to reduce the unwanted details.

Two techniques will be reviewed: a *circuit graph representation* that provide information only about the circuit's topological structure at the expense of any functional knowledge. A complementary method, *binary decision diagrams*, that hides structural details but accurately models the circuit's function is summarized. Both of these methods may be used to model a circuit's hierarchical nature very easily.

A hybrid circuit model - *gBDD or graph binary decision diagram* - which combines the positive attributes of both modelling methods will be presented. Its ability to aid the ATPG by explicit inclusion of guidance heuristics and structural information will be described.

4.2 A Review of graph techniques.

Given that an extended version of FAN will be used to discover the presence of hard faults within the logic circuit, a circuit modelling method which efficiently supports the test generation process is required. A more traditional approach is that of graphs.

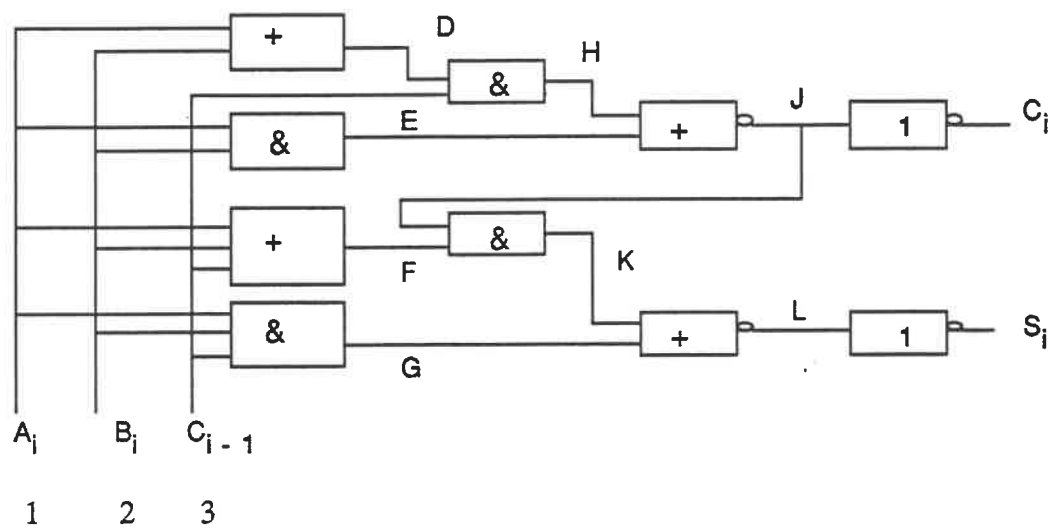


Figure 4.2.1: Full adder.

Graphs provide an elegant method for describing a circuit's structure. The full adder in Figure 4.2.1 can be described by the graph of Figure 4.2.2. Each *vertex* represents a functional logic primitive, such as a NOR or inverter logic gate, of the logic circuit. Similarly, the circuit's primary inputs and primary outputs are shown as vertices. Directed *edges*, resulting in a *directed graph* or *digraph*, link the various vertices and show the circuit's interconnectivity. A graph G, is often summarized as G(V, E) where V is the set of all vertices and E is the set of all edges. Although this is the normal case, Bell interchanges the role of V and E in his 1988 paper [Bell Taylor 88].

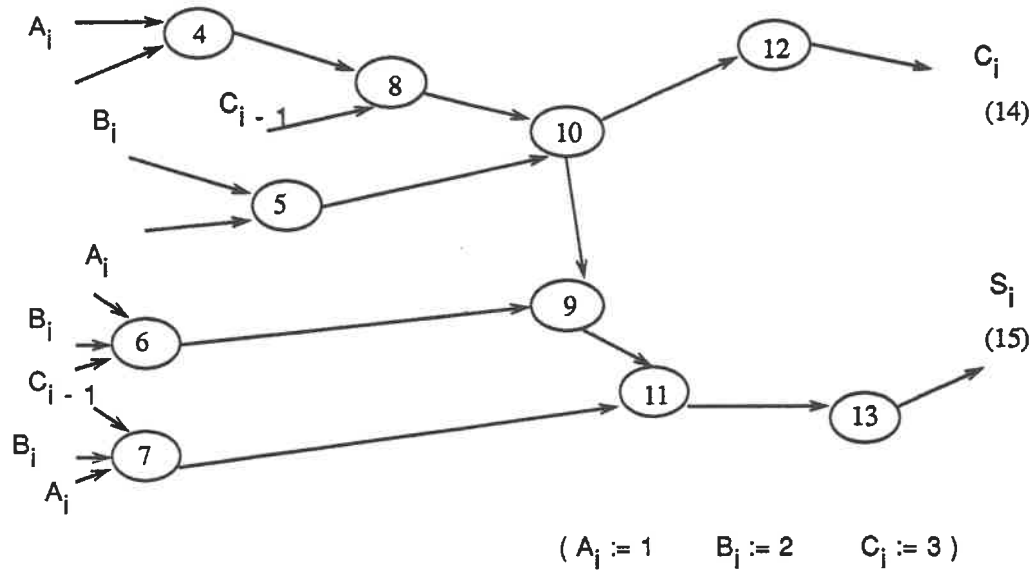


Figure 4.2.2: Full adder's graph.

Given the finite set, *{A}*, of elements contained in the UUT, such that {A} = {PIs, POs, logic primitives}, the relations between the various elements can be expressed using a matrix. Given that |{A}| = n, the $n \times n$ matrix M_R will contain a '1' (a_{ij} = 1)to indicate that there is a directed edge *e* from vertex a_i to a_j iff (if and only if) $a_i R a_j$. (A '0' indicates that there is no edge from a_i to a_j). R is the relation expressing some information about how the two vertices are connected by some *path*. The *matrix of a relation* M_R for the full adder is shown in Figure 4.2.3. It is possible to determine if a path of length n exists from an arbitrary vertex a_i to a_j by constructing $M_R n$ (a relation matrix indicating that a path of length n exists between a_i and a_j) using the relation "•" described by [Kolman Busby 84].

$$M_R^n = M_R \bullet M_R \bullet M_R \bullet \ldots \bullet M_R \qquad 4.2.1$$

The knowledge is used recursively to generate $a_i R^{\infty} a_j$ the *connectivity relation*, which indicates that there is some path from a_i to a_j. The corresponding relation matrix can be calculated from the relation 4.2.1. A more efficient technique for generating these matrices has been created. Warshall's algorithm [Warshall 62] is one such method.

$$M_R^n = M_R \cup M_R^1 \cup M_R^2 \cup \ldots \qquad 4.2.2$$

Despite the mathematical preciseness of a pure graphical representation, the lack of functional information precludes helping the automatic test pattern generator, in the case of an automated design environment, on propagating through the graph even though knowledge about the shortest path may be determined. Also, calculations using matrices tend to be exponential in the number of vertices contained by the graph. However, authors such as Bell propose using graph representations in determining TMs [Bell Taylor 88] which consider the effect of reconvergent fanout. Bell develops a methodology to detect the presence of this reconvergence and to detail information about each reconvergent path. His technique includes the ability to work with hierarchically described circuits.

4.3 A Review of binary decision diagrams (BDDs) techniques.

Akers proposed that one could replace traditional functional descriptions of logic circuits - *truth tables, Boolean equations, and Karnaugh maps* - by a concise technology implementation free description [Akers 78a] known as *binary decision diagrams*. The former have the undesirable side effect of growing at an exponential rate; they are exponential in *n* the number of variables that describe the function. This diagrammatic technique shares many of a binary decision tree's properties, however the BDDs may have more than one branch directed into it - an *in degree (id) > 1*. BDDs tend to contain a number of nodes, corresponding to the *n* variables, that grow

linearly with *n* although this can degrade to $O(2^n/n)$.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
2	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
3	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
6	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
8	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
10	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4.2.3: Full adder's relation matrix.

Figure 4.3.1 depicts the BDD for an *n-input* AND logic gate whose inputs are $\{X_1, X_2, ..., X_n\}$. Each variable is found in at least one *BDD node*; the variable is referred to as the *node variable*. The *node variable name* describes the value

assigned to this BDD node and is $\in \{0,1\}$. A node variable of 0 *activates* the *0-branch* (the left branch leaving the BDD node); a value of 1 activates the *1-branch*; X indicates that no path has been activated and that the current setting of the variable is *don't care.*

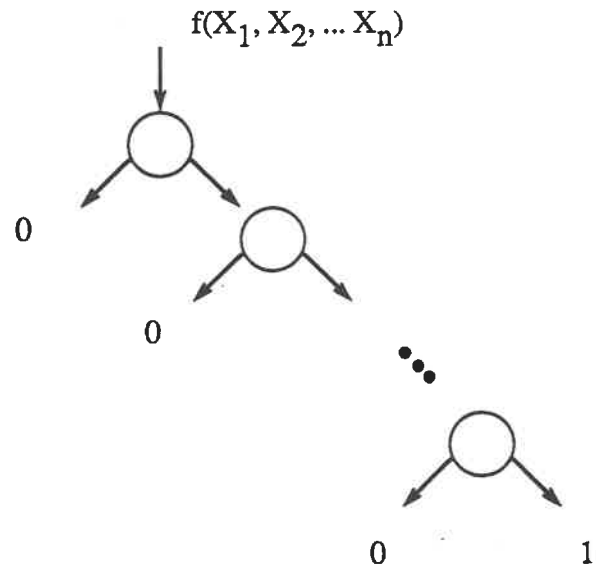


Figure 4.3.1: *n - input* AND gate's BDD.

BDD nodes having two leaves are called *exit variables*. The logic 0 and logic 1 associated with a node's leaf are the *exit value* and define the value of the evaluated function when an *activated path* - a path leading from the root of the BDD to one of the BDD's exit value exists under a given set of input conditions. Figure 4.3.2 shows a binary decision diagram for a 2 input modulo logic primitive (exclusive or gate). The activated path is indicated by the heavy line and corresponds to the input cube $\{X_1, X_2\} = \{0,1\}$. Note that the "•" indicates that *parity* is involved. For *even parity*, the function is simply the exit value; an *odd parity* requires the inversion of the exit value.

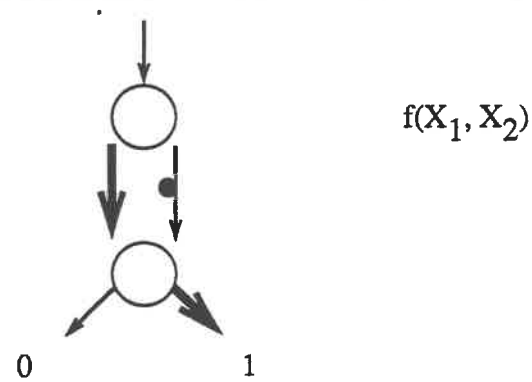


Figure 4.3.2: Activated modulo 2 BDD.

Any path that causes f to be a "1" is an *implicant* of f - thus one can generate the *sum of products (SP)* form for f by tracing all paths that go from f to a "1". The *product of sums (PS)* form is derived by selecting all the paths which go from f to a "0". Note that the implicants are not necessarily prime implicants [Akers 78a].

Binary decision diagrams are not limited to describing only combinational circuits. Akers described flip-flops (D F/F, T F/F, J-K F/F), priority encoders, shift registers, addressable latches and an *ALU* (arithmetic logic unit): some are shown in Figure 4.3.3. Thus BDDs offer a great deal of versatility in modelling a circuit's functionality. This model also supports a circuit hierarchy by using *auxiliary functions*. The replacement of a node variable by an auxiliary variable a_i, requires that the function described by a_i be evaluated before the respective *0-* or *1-branch* may be taken. Akers uses this techniques to describe a 14-input, 8 output ALU with a 35 node binary decision diagram [Akers 78b] while a truth table representation requires 2^{17} entries to provide the same information.

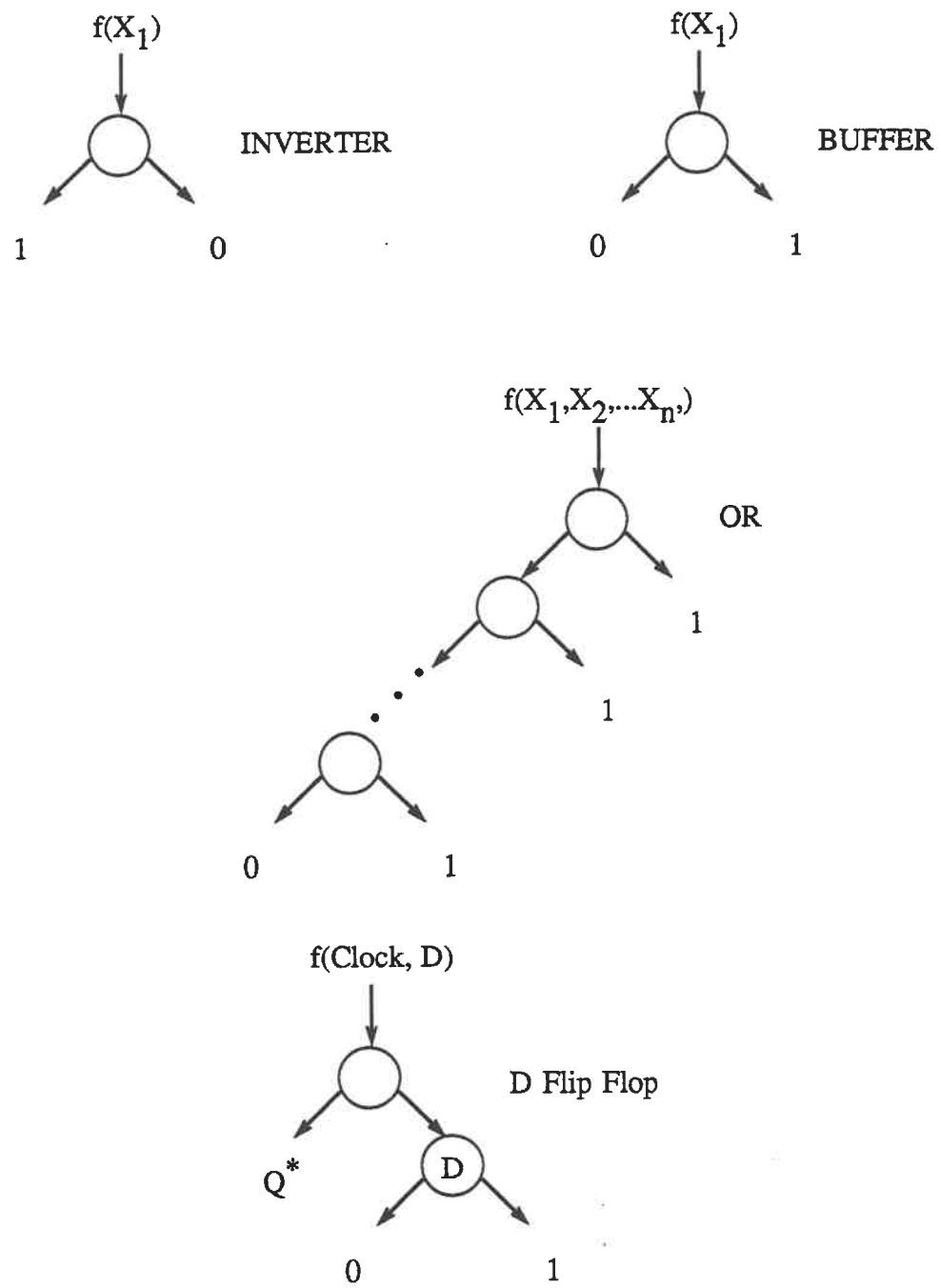


Figure 4.3.3: Typical BDDs.

BDDs can also be used to for obtaining information about the Boolean logic expressions used to describe a logic function. Branch labelling procedures can determine the number of *product terms* by counting the number of 1-exit values. *Sum terms* can be enumerated similarly from the number of 0-exit values. Akers states that similar procedures exist to:

- count the number of literals in 2 level form;
- count the number of minterms;
- count the number of maxterms.

BDDs have also been used to develop test stimulus built upon the circuit's functionality instead of the more traditional single stuck at fault model. *Functional testing* - in the sense of verifying that the device functions according to its objective specifications - may be tested using BDDs [Akers 78b]. Akers suggested that one approach to ease the testing problem is to use a *set of experiments*. An experiment is defined as a *path from f to an exit value*. Following this initial work, other researchers extended the concepts with perhaps the most definitive - to date - from a testing perspective being the work by Chang *et al.* [Chang *et al.* 86].

Functional based test highlights the two major disadvantages of the binary decision diagram circuit model. As Chang noted, implementation independent test generation can result in a varied single stuck at fault coverage quality when the different circuits are simulated using a traditional fault simulation tool. It was also necessary to create a new fault simulator to support the new fault model. Chang also noted that it is not sufficient to test that a circuit does as intended, but that no undesired functions exist due to the manufacturing process.

4.4 gBDD - Graph Binary Decision Diagrams.

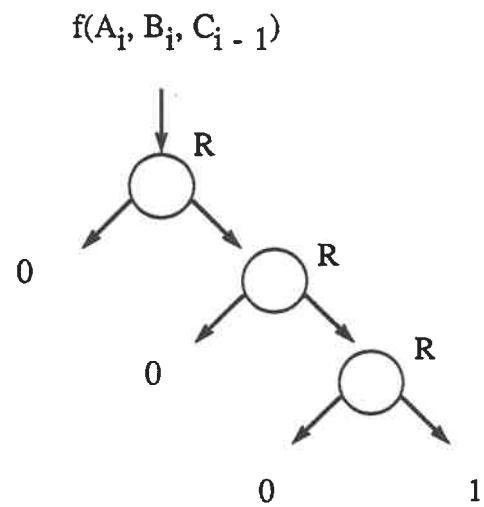
The graph and binary decision diagram circuit models have major disadvantages but ones that would effectively cancel out should a hybrid method be created. The graph lacks functional details, which are easily supplied by the BDDs; binary decision diagrams provide no structural knowledge - this can be overcome by

the graph structural modelling. A hybrid circuit representation would reinforce the positive aspects of both modelling methods. Previous researchers' work have created methods, procedures and algorithms for manipulating graphs: (1) to extract structural based testability measures [Bell Taylor 88] derived from knowledge about reconvergent fanout and path lengths; and (2) for improving fault diagnostic capabilities for the network [Russell Kime 71]. This previous research can be vertically integrated into a hybrid approach. Similar test related work, built around BDDs, developed to permit functional test generation, functional fault modelling and fault simulation procedures [Chang *et al.* 86] may also be reused. Both techniques support multiple output circuit elements and allow hierarchy. The ability to support hierarchical circuit descriptions is made more powerful by containing structural and functional knowledge down to the flattest schematic representation level (traditionally that of the gate level for ATPGs).

Thus a *graph binary decision diagram* (gBDD) circuit model was created by adding a modified BDD at each graph vertex for each of the vertex's corresponding circuit outputs since the BDD describes the output function as controlled by the setting of its input variables.

The modifications to the binary decision diagram allow the addition of propagation guidance heuristics. This permits guidance knowledge, that is a function of implementation aspects, to be directly embedded into the circuit representation/ For an *n-input* AND gate, the *modified BDD* [Stannard Kaminska 89b] is shown in Figure 4.4.1. Intuitively, the test engineer knows that all the AND gate's inputs, except for that with the error signal, need to be set to a logic '1' before the error will be propagated to the AND gate's output: this corresponds to choosing the *1-branch* or the right branch. This data is automatically inserted into the mBDD structure at the time of the gBDD's creation. This mBDD structure, at the primitive level, lends itself nicely to several heuristics during the justification phase: the mBDD's nodes can be explicitly ordered by decreasing difficulty of setting a logic '1' on the input variables by using the COP C_1 TM while implicitly stating the C_0s. Other orderings

could also be used, however since TMs do not always correlate with the actual difficulty in detecting a fault, it was arbitrarily decided to choose the latter and hope that, on average, the tradition of trying to justify the hardest net value first will allow quick identification of conflicts and backtrack situations.



Modified BDD for Graph Node 6

Figure 4.4.1: Modified BDD for AND gate.

The graph binary decision diagram approach requires that each *directed edge* corresponding to a *unique* logic element output have a modified BDD "attached" to it. Also, a *graph node's* (GN) *out degree* is no longer the traditional number of directed edges emanating from the corresponding vertex; it is now the quantity of unique logic element primary outputs or *GN_POs*. For the purposes of this thesis, only single output logic primitives have been considered since handling multiple outputs is an implementation issue and not central to this research.

Conceptually, the 3 input AND gate's modified binary decision diagram is added to the appropriate graph node edge such as at graph node 6 in Figure 4.4.2. This was implemented in the "C" language version using data structures for the

graph nodes which included a vector of pointers for every unique (i.e. one per logic element output) mBDD.

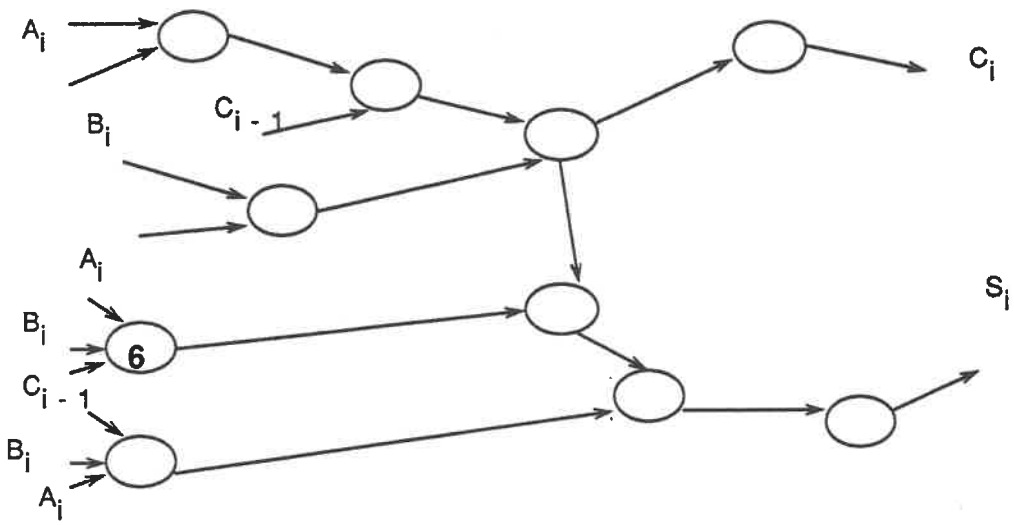


Figure 4.4.2: gBDD for full adder circuit.

Figure 4.4.3 demonstrates simplified versions of the graph node and modified binary decision diagram data structures used in modelling the unit under test (UUT). Every mBDD node has a pointer to its corresponding input variable.

Each of the graph and the binary decision's advantages are reinforced by the this hybrid circuit modelling method. The net advantages are:

- Functional and structural information;
- Supports hierarchy;
- Sequential logic elements supported;
- Structural (that is the single stuck at fault model) and functional faults may be used;

- Can integrate the work of previous researchers;
- Can build function verification tests once the blocks and their functionality are known - following the objective specification for the UUT.

```
GRAPH_Node{
        PI_data_Vector[GN_PI_quantity];
        PO_data_Vector[GN_PO_quantity];
        PI_to_mBDD_node_pointers[GN_PI_quantity];
        PO_to_mBDD_Root_pointers[GN_PO_quantity];
        GN_identification_Data_Structure;
        GN_Simulation_Data_Structure;
        GN_Fault_Sim_Data_Structure;
}
mBDD_Node{
        GN_PI_pointer;
        Value;
        Propagation_Rules;
        mBDD_identification_Data_Structure;
        mBDD_Branch_pointers[Up, Down, Left, Right, Previous]
}
```

Figure 4.4.3: Graph node and mBDD node data structures.

4.5 Summary.

Two traditional circuit models were reviewed and their attributes were explored briefly: the graph model with its lack of functional data, and the binary decsion diagram which compensates for the lack of functional information at the expense os structural knowledge. A new circuit model, graph binary decision diagrams consisting of a hybrid of these two techniques, was introduced. This circuit model was required to allow HUB to detect the presence of hard to test faults and support the various phases of this algorithm.

5.0 Detection of Hard Faults using HUB.

5.1 Introduction to budgetary constraints.

Despite the use of *testability measures* and the other techniques reviewed in Chapter 2, there does not appear to be a valid method to detect the presence of *hard to test faults* (HFs) in a logic circuit. The use of an algorithmic automatic test pattern generator, while being the definitive method for finding redundant faults, is generally considered to be too unrealistic to be considered a valid approach when large number of UUT_PIs are encountered. Thus for larger circuits, it is unacceptable to run an ATPG with unlimited backtracks allowed and no time limit. Detecting HFs with an algorithm is also a function of the choice of algorithm, heuristics and the search strategy [Min Rogers 89].

Additional problems associated with using ATPGs and TMs to detect HFs may also surface:

- the backtrack's cause is not known;
- the actual elements involved in the aborted test generation effort are not known. That is the structural and functional information is not available;
- the phase that initiated a backtrack is unidentified;
- the ATPG or TM calculation effort is essentially lost since most tools need to be rerun for the whole circuit once modifications have been selected and implemented.

Therefore it was decided that another approach was required, one that allows a more practical attack on the problem - that is the identification of the HF and, hopefully providing some guiding information that would be useful to the design team in determining a solution set to correct the HF's root cause. Thus a new set of metrics, founded upon the concepts of circuit budget constraints [Stannard Kaminska 88a], was developed; they were suggested as a means by which the loop between the designer's circuit knowledge and the chip's inherent testability could be closed.

Usually, a project has a budget for all engineering related costs - usually referred to as *Non Recurring Engineering* (NRE) costs since they should happen only once per product - for the phases of a new product's introduction. A typical integrated circuit design process flow is presented in Figure 5.1.1. There are many variations of this theme, often catering to an organization's internal requirements. Henckels presented a graph showing the test NRE component for test generation, relative to the total design NRE [Henckels 88a]. As circuits continue to become more complex, the tendency appears that test NRE may approach 50%. Henckels also indicated that ASIC (application specific integrated circuits) do not appear to be heavy SCAN method users due a variety of reasons. In today's marketplace, cost effectiveness and time to market are key ingredients to a product's success. Thus, budget considerations will be applied to the UUT in order to couple a circuit's testability to the reality. Budgets, time, and costs tend to be known items for the company due to its past experience in project management and its understanding of its market. The feasibility study will allow knowledge of whether the project is worth the effort and is potentially profitable; the prior lack of testability for a class of circuit will probably result in the project stopping before a major effort and difficulties are expended.

Before explaining how the system HUB (Hard fault detection Using Budget constraints) monitors costs during its hard fault identification efforts, it is necessary to introduce three circuit compartementalization concepts: (1) the *pdcf* and its associated *pdcf fault*; (2) the *graph node (GN)*; and (3) the *UUT*. The pdcf (*primitive D cube of failure)* describes the input condition to a combinational logic element in order to sensitize the element for the presence of a single stuck at fault. In general, there are $n+2$ pdcfs for an n - input logic element. The pdcf is able to detect a fault, and sometimes two faults (always under the assumption of a single fault at any given time) and these enumerated faults have been labelled as pdcf faults.

The *GN* corresponds to a single logic element primitive and has $n + 2$ pdcfs associated with it in the case of the single output logic primitive as is the case for all the circuits examined during this thesis.

The *UUT*, or unit under test, is the whole circuit as viewed from the highest level of hierarchy. The UUT is composed of all the internal building components (AND gates etc.) which are represented in the gBDD circuit model by an associated GN. All of these items have costs and budgets assigned to them and the increasing degree of resolution as one procedes from the UUT level to the pdcf level allows for the accounting to be controlled at various degrees. The UUT's budget is static, but all costs and budgets are dynamic within the global UUT constraints.

HUB monitors costs at three separate levels: (1) per *pdcf fault*, the fault(s) associated with the error sensitization of the targeted *GN* for the graph node's fault dictionary; (2) per *graph node*; and (3) for the *UUT*. These costs are then compared dynamically against a set of static and dynamic budgets and consist of:

- propagation cost (CPU - central processing unit - time);
- justification cost;
- implication (simulation) cost;
- backtrace cost;
- backtrack cost;
- total cost - the total of the above 5 items.

The UUT's *budget* is defined by the program's user and is *static* for the duration of HUB's run time; this budget is distributed over the UUT's fault set. As hard fault detection program proceeds from graph node to graph node, local dynamic budgets are created as a function of the potentially detectable faults at that GN or the node under test (NUT). At the pdcf level, a budget is created for the number of potential faults that will be detected if the pdcf can be justified and the error propagated to at least one UUT_PO

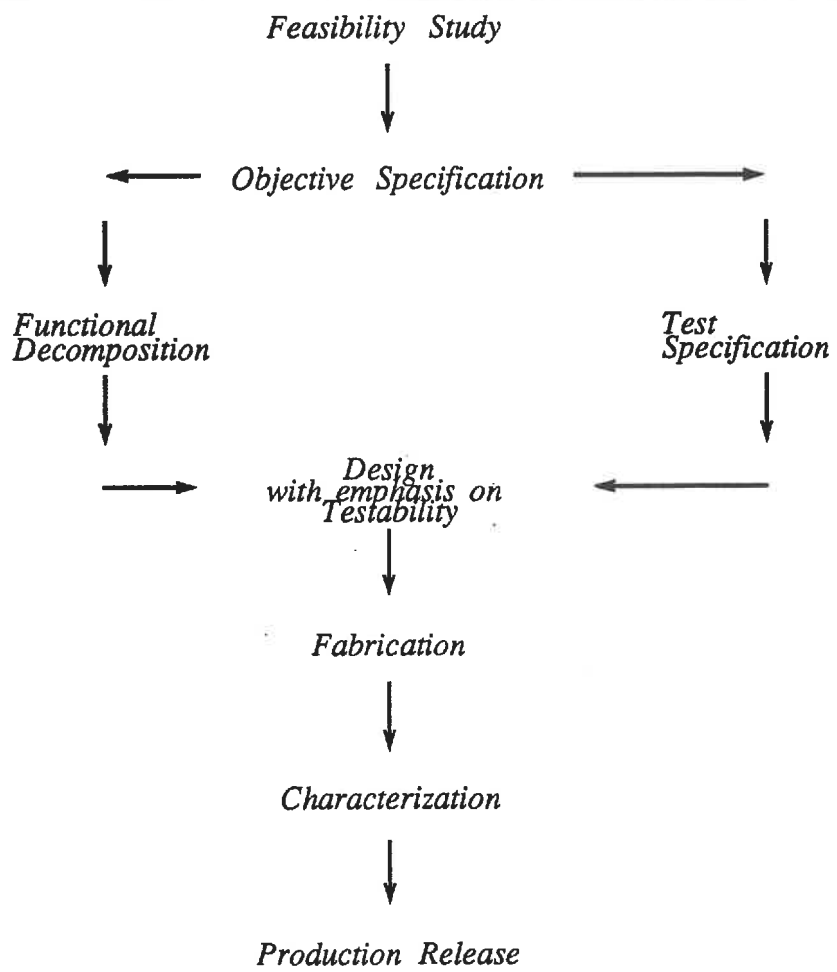


Figure 5.1.1: Design phases.

The budgets must be respected, given the algorithm's and the computer's ability to resolve time differences; thus each major ATPG phase charges its execution cost (CPU time) for each invocation. Therefore all pdcfs must remain within their budget constraints for the total pdcf time while simultaneously requiring that the GNs and the UUT also remain within their respective budgets. The UUT's budget takes precedence over the individual GN's budget which in turn is of higher ordering than the individual pdcfs. The order of precedence for the various budgets are summarized in Figure 5.1.2.

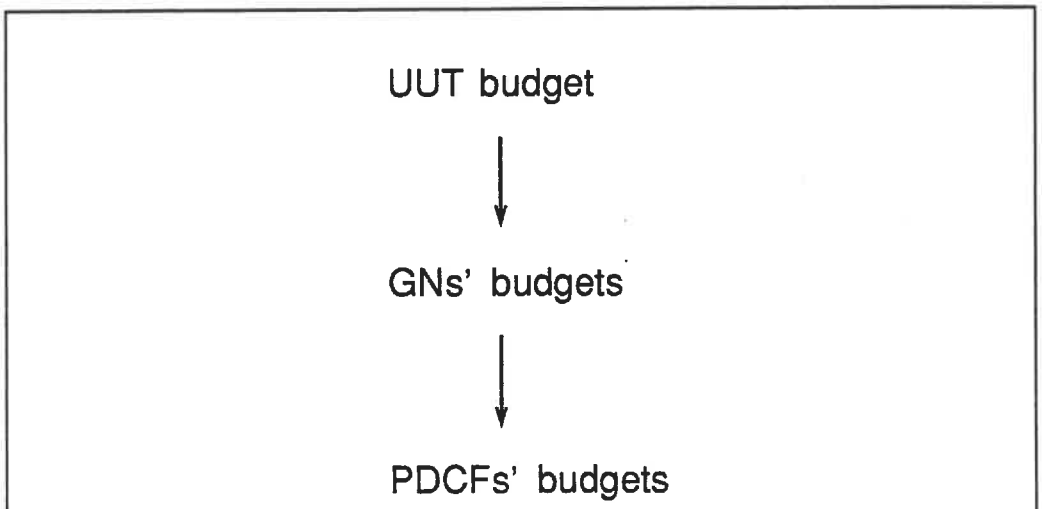


Figure 5.1.2: Budget importance.

Failure to meet a budget has repercussions depending upon which budget is not respected. Overspending of a *pdcf budget* results in *potential HFs*; if subsequent test vectors generated for the other pdcfs of other graph nodes are not capable of detecting these faults during a fault simulation phase, then these potential hard faults become HFs. HUB will continue with any remaining pdcfs for this same graph node provided that the GN's and circuit's (UUT) budgets are respected. Should the graph node have no pdcfs remaining, HUB will select the next GN to become the *Node Under Test* (NUT) providing resources (time) at and circuit levels permit. Thus, as one proceeds from the top to the bottom, of Figure 5.1.2, the implications of being over budget have a more localised effect.

Overspending at the GN level results in all undetected faults at the GN becoming potential HFs which are subject to the same fate as stated above. HUB will continue with another GN should any remain and the UUT's budget be respected. Overspending at the UUT level causes the immediate classification of remaining faults in the valid fault dictionary as hard faults. A definition of hard faults based upon

cost constraints follows from this hierarchy. Figure 5.1.3 presents some of the formulae used to administer the costs and budgets.

Hard Fault	•	a fault that can not be detected within the cost constraints and that is contained in the fault dictionary. The costs are those imposed by time, backtracks limits, and the exhaustive search nature of the underlying algorithm.

Fault_count = Desired_fault_coverage * Total_fault_count

$$\textbf{Resource_per_fault} = \frac{\textbf{Total_resources\$}}{\textbf{Fault_count}}$$

PDCF_budget = PDCF_faults * Resource_per_fault

GN_budget = GN_faults * Resource_per_fault

PDCF_costs = Σ(ATPG_phases' costs for PDCF)

GN_costs = Σ(GN's PDCF costs)

UUT_cost = Σ(GN costs)

Figure 5.1.3: Budget formulae.

5.2 The HUB algorithm.

HUB strives to identify hard to test faults by using a deterministic algorithmic test generator controlled by sets of cost constraints and guided by a mixture of traditional heuristics (COP based testability measures) and heuristics embedded

into the gBDD circuit description. The algorithm is a modified version of Fujiwara's FAN algorithm with several differences in its implementation and methodology. The test generator is augmented by a fault simulator, which currently supports only the single stuck at fault model, that helps reduce the test generation effort; these will be described in more detail. Before HUB can be invoked, the circuit must be converted from its original description, in a TEGAS like language, to a file containing the information about the gBDD structure. It is during this phase that the COP TMs are calculated; this data is also stored in the file. These tasks were consigned to a separate phase specifically to reduced the cost of running the test generator: since all the data is static during the automatic test generation, there is little to be gained by repeating these tasks needlessly each time that the ATPG is used for the same circuit. Figure 5.2.1 indicates this data flow.

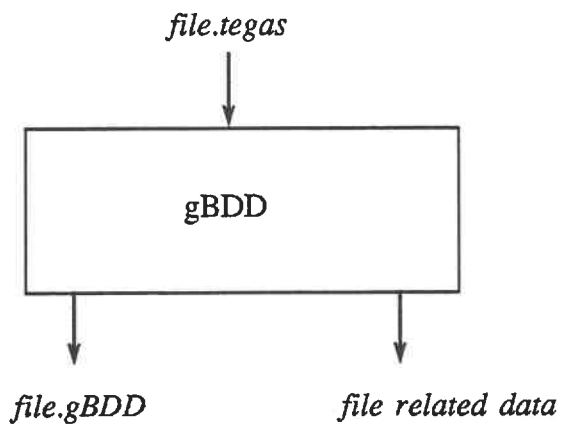


Figure 5.2.1: Creation of gBDD circuit.

The ATPG based approach was adopted since teatability measures have yielded the desired level of confidence and since it has been suggested that it is important to link the testability to the test generation process [Bell Taylor 88]. Algorithms also provide redundant fault detection in the case of an exhaustive search. The ATPG, if controlled, should provide a wealth of insight through its stored information generated during test generation about the HF's cause. Use of HUB requires that its user provide a set of 3 cost related constraints that will be used to

define and detect a HF. Specifically, the program requires:

- the minimum desired fault coverage [0.0% ... 100%];
- the total budget for the UUT;
- the backtrack limit per target fault.

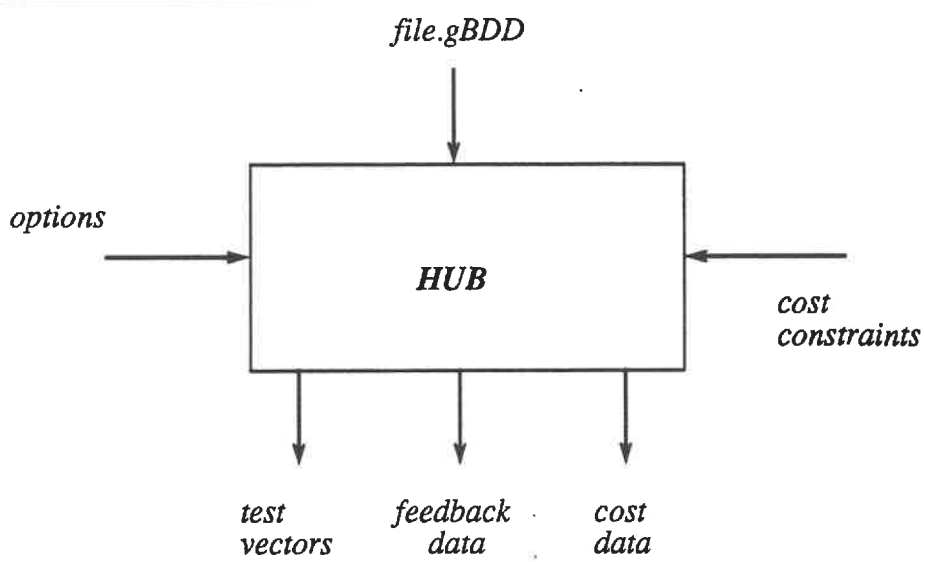


Figure 5.2.2: HUB's inputs and outputs.

All 3 data are considered to be important: the traditional backtrack limit was kept since prior researchers indicated that there is some merit in specifying a reasonable ceiling, perhaps 10 backtracks [Fujiwara 85][Ivanov 85]. The minimum fault coverage provides the user with the ability to accept ''poorer'' test coverage as defined by the metric. In the eventuality of tight budget constraints the user can try to pick this fault coverage to arrive at the desired point on the typical fault coverage curve; a typical fault coverage curve is shown in Figure 5.2.3. The budget refers to the CPU (computer time) resource that will be allocated to the UUT. This resource is distributed evenly over the circuit on a *per fault* basis: that is the time will be amortized over the *total potential fault quantity to be detected.* This potential fault volume will be a function of whether the faults were collapsed or not. While for this thesis the resource is uniformly distributed, weighting of the budget would provide the ability for the design team to emphasize the important circuitry, in their opinion,

instead of based upon grouping of potential faults.

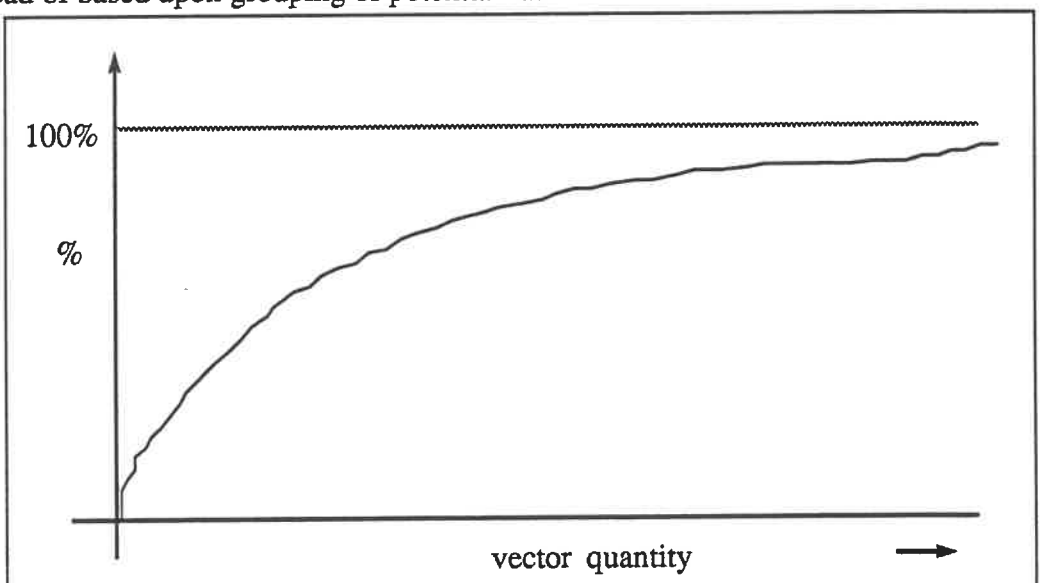


Figure 5.2.3: Fault coverage curve.

A budget of $0.00 provides the immediate and obvious result that the UUT is untestable for all the faults. An unlimited budget reverts to the backtrack limit criteria or an exhaustive search of the decision space depending upon the backtrack limit's magnitude, the size of the decision space, and the efficiency of the underlying algorithm and the heuristics.

The pseudo code for HUB, presented in Figure 5.2.4, represents the major functional blocks executed by the algorithm. Initially, the circuit is read into the computer's memory using a linear algorithm where it is stored as the internal gBDD circuit model. The user supplied constraints are obtained and the reference metrics are calculated: the total valid potential fault quantity using fault collapsing should this option be specified; the *CPU* resource allocation per fault is then determined knowing the fault dictionary's volume. Dynamic metrics such as the *pdcf budget* and the *GN budget* are calculated at each GN as required and only if there are undetected faults remaining at the targeted GN. If the next *pdcf* sensitizes faults already detected, HUB looks for the next possible pdcf which has undetected faults associated with it; when no pdcfs remain, HUB proceeds to the next GN among the remaining valid

choices.

```
Restore_gBDD(circuit);

if (fault collapsing required)

        Fault_Collapse(circuit);

Count_Potential_Faults(circuit);

Query_User();

Define_Budget_Metrics();

HUB_atpg(circuit);
```

Figure 5.2.4: HUB's pseudo code.

The ATPG process is the most important and time consuming portion of HUB. It is this function, described by the pseudo code shown in Figure 5.2.5, that risks have exponential run times when faced by hard to test circuitry. Attempts have been made to make the automatic test generator an efficient one; thus FAN serves as its basis and heuristics and implementation related features have been added above the basic concepts of directing HUB by cost controls. The primary goal was not to reproduce an ATPG, thus it was not considered essential that the implementation be the most efficient available nor the most compact coding and sophisticated use of heuristics.

```
while (Faults remain in UUT){
    Pick a fault.
    Sensitize the fault.
    while (No test yet & Not exceeded backtracks){
        Implication.
        if (Error at UUT PO){
            if (No unjustified BOUND lines){
                Justify FREE and HEAD lines.
                Simulate for other faults detected by this test.
            } else BackTracE.
        }
        else if (Error not at UUT PO){
            else if (D Frontier = 1 gate){
                Unique Sensitization.
                BackTracE.
            }
            else  BacKtracK
        }
    }
}
```

Figure 5.2.5: HUB ATPG's pseudo code.

Hard faults are only those faults which were not detectable from the set of targeted faults. If a 0% fault coverage target is chosen, there are no HFs although there is 100% undetectable faults. However if a 100% fault coverage value is chosen but 0 resources allocated, all the undetected faults are hard faults even though they may be detectable.

Unlike FAN, no multiple backtrace phase was generated since this was thought to be an unnecessary phase and since initial effort suggested that there was no clear advantage in implementing it. [Marlett 89] and [Min Rogers 89] would appear to support this decision.

The time cost of each of the 5 important phases (of the FAN based test generation algorithm) are timed and these times are charged to the respective account at the pdcf (lowest), graph node, and unit under test (highest) levels. The total effort required at each level is monitored for adherence to the respective budgets. Costs are accounted for using the data structure shown in Figure 5.2.6. At

```
Accounting->Propagation_time
Accounting->Justification_time
Accounting->Implication_time
Accounting->BacktracE_time
Accounting->BacKTracK_time
Accounting->Total_time
Accounting->Budgeted_time
```

Figure 5.2.6: HUB Cost Accounting Data Structure.

the end of each pdcf's ATPG effort, the pdcf accounting structure data is used to update that of the NUT's (at the graph node level) and that of the UUT. The pdcf accounting structure is then re-initialized. Upon completing the NUT's ATPG work, the GN and the pdcf's accounting structures are re-initialized. Note that the budget is the product of the *resource_per_fault* and the *fault_quantity*. For the UUT, this is simply the user specified budget constraint value.

Although this was not implemented, one could weight the various cost groups at the graph node and the pdcf level. This would allow more resources to be allocated to critical faults and circuitry.

When HUB chooses a path for error propagation, if there is more than one gate in the D Frontier, by performing an X-path check [Goel 81]. Essentially this starts with the fanout branch having the largest observability (all values are kept separately instead of heuristically setting all fanout branches with the same value) and that is not blocked before reaching an UUT_PO. Should none remain, a backtrack condition occurs.

5.3 Important HUB attributes.

HUB has some features that are discussed in futher detail in the following paragraphs. (1) The algorithm's progression through the circuit is controlled by the list of graph nodes and not by the list of faults as is typically the case. (2) Sensitization of the target fault(s) at the GN is dictated by the use of information contained in a *pdcf personality file* and which currently supports only the single stuck at fault model and not by faulting every node with each of the single stuck at faults. (3) A provision for simplified method (heuristically based) of single stuck at fault model fault collapsing is provided. Also provided are an interactive mode for selective ATPG or HF identification that is controlled by specifying this option upon program startup with the number of the desired GN and one of its pdcfs. (4) HUB produces a wealth of information that is output to the user for use in solving hard faults, for program devlopment purposes, and for educational purposes. (5) Also included, but not currently used, is the provision for the introduction of hierarchy at a future date. Each of these attributes is explained below. HUB also retains a wealth of other information acquired during the ATPG phase; this retained data will be briefly introduced but, due their planned usage at a future time, no in depth details will be provided.

5.3.1 ATPG control by Graph Node (GN).

Typically, a fault dictionary is created for the circuit and this fault dictionary contains *unranked faults* [Shen 85]. Commercial simulators such as SIMUCAD's *SILOS II*™ do not explain or make readily available any information on how this fault dictionary is created, its structure, how it is accessed, or how it is maintained during simulation. One supposes that, upon creating this fault dictionary, faults are accessed in a sequential order and that one might find the next targeted fault in a totally different region of the unit under test. A version of PODEM, used at École Polytechnique, seems to create a fault dictionary dictated by the order in which the circuit is processed starting from the UUT's PIs.

HUB functions by attempting to complete the test generation at a *targeted Graph Node* for all the faults required to be detected at this GN before selecting another graph node. If the fault simulator option was invoked, the additional faults detected, which must occur for GNs other than the current GN, will be identified at their respective graph node.

The gBDD circuit model does not attempt to flatten the circuit by element depth. Rather the circuit is stored starting with all the UUT_PIs and then is tried in an order determined by the way that the gBDD is traversed from the UUT_PIs towards the UUT_POs in a sequence dictated by the circuit's structure and a depth first algorithm; pointers to each graph node are stored in a simple linked list shown in Figure 5.3.1.1. A list of graph node pointers, each pointer is represented by a rectangle in Figure 5.3.1.1, is created when the gBDD circuit model is generated. This list is ordered starting with pointers to the graph nodes of the circuit's primary inputs and corresponds to the entries 0 ... n in the pointer list; these pointers are strictly for primary inputs. Graph node pointers $n+1$ through m point to graph nodes which represent either logic elements or a primary output for the circuit. Although this list of pointers is guaranteed to start with the UUT_PIs' pointers after which one can find UUT_PO pointers interspersed with those of the logic elements. The presence of an UUT_PO pointer may indicate a cone of influence but this is not guaranteed since

the UUT_PI ordering is a direct consequence of the circuit's netlist and may not reflect any cone of influence; their ordering is only a function of the individual (or algorithm) who created the original circuit netlist.

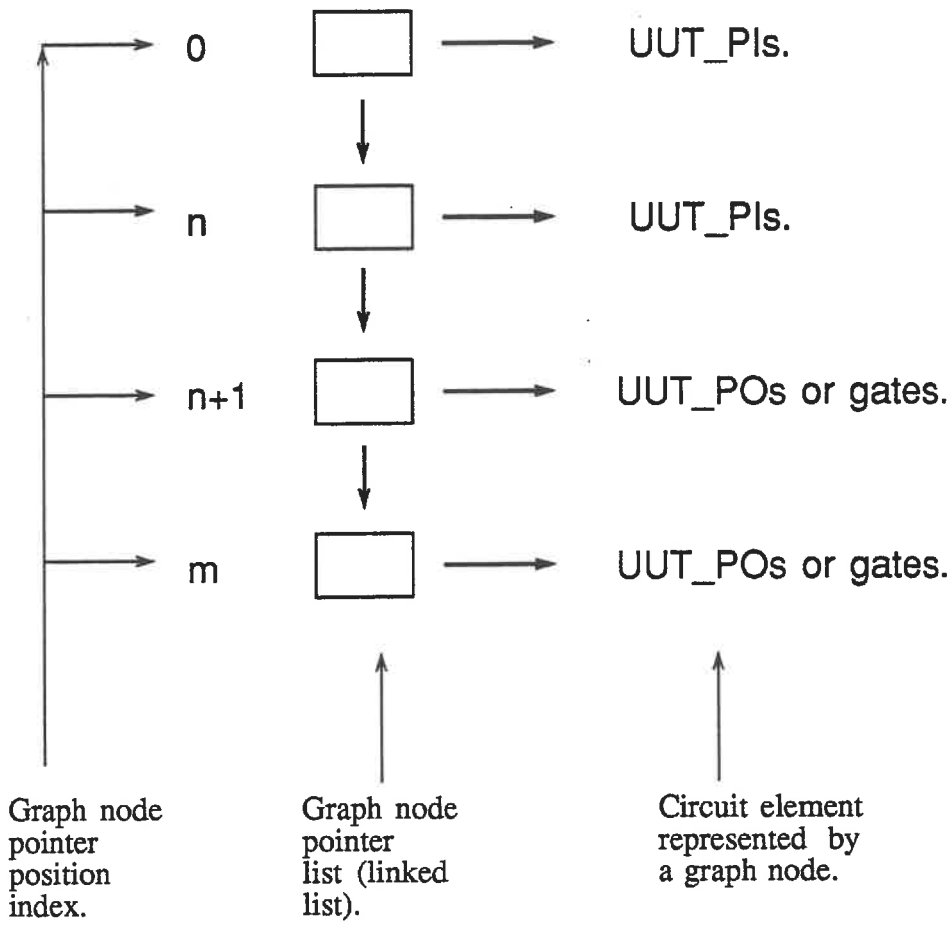


Figure 5.3.1.1: HUB's graph node pointer list.

HUB proceeds from GN to GN starting at the first graph node which is not an UUT_PI or an UUT_PO. For this research, all of the UUT_PIs' graph nodes are skipped since the fault collapsing routine and the ATPG assumes that the faults possible on these inputs will be equivalent to other faults on the nets to which they are attached. This assumption can be incorrect when the primary inputs fanout.

UUT_POs are also skipped since it is assumed that their faults will be equivalent to those of the logic element's output controlling them.

5.3.2 pdcf Personality File.

Each unique logic primitive has a *pdcf personality file* associated with it. This file, for this thesis, contains the *primitive d cube of failure* associated with the $n + 2$ single stuck at faults which are detectable at the gate (with the notable exceptions of the modulo 2 function with $2n + 2$ single stuck at faults and the inverter/buffer functions with 2 single stuck at faults). Additional information describes how many and which faults are detectable by this pdcf. This approach was used so that the flexibility of using precomputed test stimulus was permitted. There is a desire to cut test costs by reusing previous work and this was designed into HUB from the beginning to allow for its use in hierarchical test generation. An example of the pdcf personality file is shown in Figure 5.3.2.1. Information includes the number of faults that can be detected by all the pdcfs contained within the file, the number of pdcfs in the personality file, and which faults cannot be resolved due to fault dominance. Each pdcf which shows the error free output value, the output value when a fault is present, the number of faults and which faults are detectable by the pdcf: "*i*1/1" refers to input 1 stuck at 1 while the "*o*" indicates an output node.

```
4 faults
2 inputs
i @ 0
3 pdcfs
0 1  0  d  : 2 i1/1,      o1/1;
1 0  0  d  : 2      i2/1, o1/1;
1 1  1  D  : 1            o1/0;
```

Figure 5.3.2.1: 2-input AND gate's pdcf personality file.

While a graph node is within budget, HUB is controlled by this pdcf file. The file, as shown, could have its fault dictionary ranked according to some predetermined importance: this ranking could be by consideration of the relative probability that this fault will occur compared to the remaining faults or, as is the case above, by the number of potential faults detectable. Thus pdcf personality files provide the ability of allowing further control of the ATPG process external to the underlying FAN based algorithm.

By using an appropriate fault identification method, independent of the underlying fault model, the pdcf file would permit precomputed test vectors to be used; these vectors could be built for the desired fault model. Obviously, the implementation will require changes to permit vector sequences however adding this feature was not considered since it is not within the principle scope of this thesis. However, the author would probably suggest a technique such as Marlett's EBT (extended time backtrace) [Marlett 86].

5.3.3 Fault model, fault collapsing and simulation.

HUB uses the single stuck at fault model; all fault quantity calculations, fault collapsing, and fault simulation are constructed under this assumption. (This was a question of time efficiency since the desire was to make these features independent of the fault model employed, but this will be left as a research subject for others.) The UUT fault dictionary's size is dictated, when no fault collapsing is desired, by the number of faults in the pdcf personality files; this is determined by assuming $n + 2$ single stuck at faults per logic gate (except the *exclusive OR/NOR* functions with $2n + 2$ faults and *inverters/buffers* with 2 single stuck at faults) given an *n-input* logic function. This $n + 2$ quantity is based upon *fault dominance* principles [Miczo 86].

One method of calculating the circuit fault dictionary's size is to simply assume that every net in the UUT can be faulted with a stuck at one and stuck at zero; thus for k nets, one would have $2k$ potential faults. This would be an upper bound and in general should be subject to reduction due to the ability to *fault collapse* given the circuit's topological nature. At a local level, that is at an individual gate

having $n + 2$ faults, one can typically reduce the fault quantity to $n + 1$ faults due to the ability to generate tests which can resolve between 2 specific faults. *Fault equivalence* and *fault dominance* can be used to further reduce the resolvable fault quantity across the entire UUT. Judging by the comments that different fault simulators using the same circuit and test patterns can yield a variety of fault coverage values, it would appear as if the fault dictionary of single stuck at faults is far from being clearly defined.

HUB provides for 2 operation modes: (1) no fault collapsing requested except for the inherent fault reduction due to the pdcf personality files and ignoring UUT_PIs and UUT_POs; and (2) fault reduction based upon the following criteria:

- the faults on logic gate's outputs are subtracted if the net to which it is attached can detect these faults. This is demonstrated in Figure 5.3.3.1;

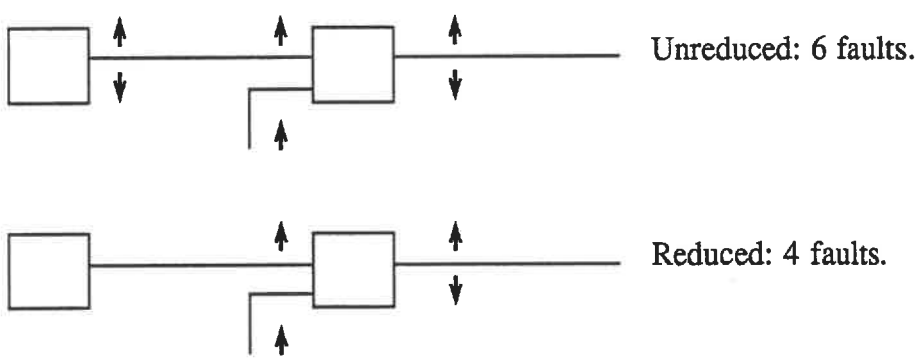


Figure 5.3.3.1: Fault collapsing.

- the faults on logic gate's outputs are subtracted if the net to which it is attached can not be resolved (detected) at the destination logic element's input;
- the faults on a fanout stem are subtracted if the fanout branches to which it is attached can detect these faults singularly or collectively;
- the faults on a fanout stem are subtracted if the fanout branches to which it is attached cannot be detected collectively.

A simple single stuck at fault simulator was created by a student under the

limiting case of combinational only circuits. Once the test vector has been generated, the fault simulator takes the values at the UUT_PIs and works towards the UUT_POs building lists of faults detected. Figure 5.3.3.2 indicates the underlying simulation philosophy which is essentially a list processing mechanism. This technique was adopted primarily to (1) aid the development process and to generate vectors with a list of fault attached for use in building new pdcf personality files, since pdcfs have a list of faults associated with them. This is also a simple intuitive approach although limited to combinational logic circuits only. The use of list was also an attempt to decouple the fault model from the fault simulation routine although no effort was spent on this line of research. The fault model decoupling concept was not successful since it was necessary to use a simple mechanism, at each graph node, to indicate that a given fault had already been detected: thus bits within a vector were used to indicate that a fault had been detected, resulting in a method equivalent to *stuck at n pin faults* [Chang *et al.* 86], instead of a true list. This plus some other implementation aspects has resulted in the simulator being dedicated as a single stuck at fault simulator.

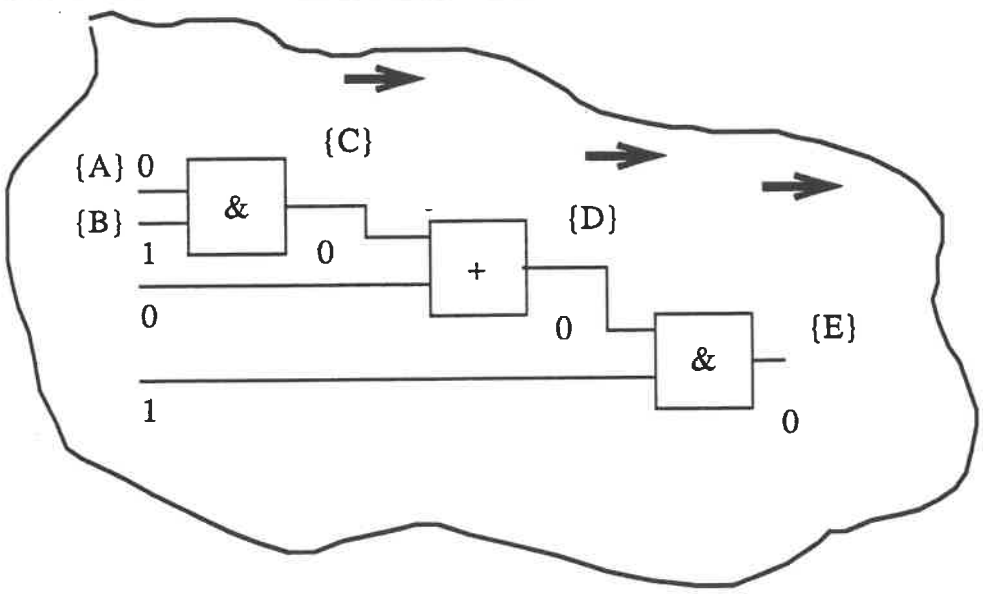


Figure 5.3.3.2: Fault simulation using lists.

Fault simulation is performed using a breadth first algorithm which constructs concatenated fault list using the information about logic elements blocking conditions. Each list is reduced by removing redundant entries and memory size is controlled by dropping lists which can not be propagated to an UUT_PO. An indepth analysis of the fault simulation routine will not be undertaken at this time, and will be the subject of a separate report.

In the example of Figure 5.3.3.2, the boolean values represent the nets' values for the fault free circuit. The list {A} will be propagated onto the output of the first (from the left) AND gate's output while list {B} will be blocked. List {C} is formed by concatenating any faults, as yet undetected, which are potentially detectable at this AND gate to the list {A}. List {D} is formed by repeating this operation at the OR gate. If a sensitized path exists from the second AND gate to a UUT_PO, then list {E} (plus any other non redundant concatenations) will represent the list of faults detected by the input vector.

A file containing the detected fault list and the quantity of faults detected by each test vector is produced when the fault simulator option of HUB is invoked. For the full adder shown above, one would have the data shown in Figure 5.3.3.3, which is a partial sample of the fault simulator's output file.

5.3.4 Interactive Mode and Information Feedback.

To permit the user to retry specific HFs, HUB has the ability to choose to attempt ATPG at a specific graph node for one of the GN's pdcfs. By selecting this interactive mode and simultaneously invoking the verbose option, HUB provide information on its progress as it tries to find a test vector; this provides information about the reasons that conflicts occur during the error propagation, net justification, and net implication phases. This was intended to aid the development process and provide knowledge about why HUB is not able to detect a fault.

```
                SIMULATION (reduit)

Date : Mon Nov  6 16:55:31 1989
Fichier d'entree : fadd.bdd
Nombre de defaut : 38  Nombre  de defaut reduit : 22


Vecteur d'entree : 101

    Liste :
        D  : i1/0
        J  : i1/0
        CARRY  : o1/0
    Liste :
        K  : i2/1
        G  : i2/1
        SUM  : o1/1

    Nombre de nouveaux defauts : 6


Vecteur d'entree : 011

    Liste :
        D  : i2/0
    Liste :
        G  : i1/1

    Nombre de nouveaux defauts : 2
```

Figure 5.3.3.3: Fault simulation sample output data.

HUB provides feedback about its performance on a per pdcf basis, a per graph node basis and for the UUT. The type of information provided is listed below and is intended to explain how the hard fault identifier expends its time and effort:

- total backtrack quantity;
- backtrack quantity for each major ATPG phase;
- total cost expenditures;

- individual cost expenditures per major ATPG phase;
- resource allocation information;
- fault target quantities;
- the identification number for the graph node causing the backtrack, if it can be clearly assigned;
- the primary and secondary phases (propagation, justification, implication, other) causing the backtrack;
- if an exhaustive search has occurred, this will be indicated;
- a list of rules violated due a conflict.

The justification for this information is that it is not sufficient to state that a fault cannot be detected due to an aborted process or due to HUB's heuristics; one must have information about the root causes in order to permit the design team to effect the necessary and required changes. Presently this information is generated in an ASCII format to permits its manipulation by UNIX™ utilities (*grep* for example) and to allow the data to be easily imported into sophisticated data analysis tools such as *BBN's RS/1*™[BBN Software Products Corporation].

Other information is generated and kept by HUB although not currently employed for this research. HUB indicates at each graph node the causes of why its GN_POs and GN_PIs have been set; this is by the identifying graph node number and the ATPG phase. It was intended that, given a means to couple this circuit model to the schematic representation, the designer could visualize the test generation process in order to understand how he might modify his circuit in order to remove any hard faults. HUB also records the number of failures in setting an GN_PO to a zero and to a one in order to provide a list of troublesome graph nodes without this integration to the schematic environment.

5.3.5 Other HUB features.

There are some other features embedded into HUB, although they might not

be used to their fullest potential: (1) the observability (from COP) at a fanout stem is not simply the minimum as is typical with COP, but the mean of all the fanout branches; (2) all the fanout branches' observability values are kept; (3) the fanout destinations are ordered according to COP observability values; (4) each graph node's output records the quantity of zeros and ones that were not justifiable, this was based on discussions with Marlett at the 1989 IEEE VLSI Test Workshop; (5) the multiple backtrace option was written and tried with smaller circuits, but was not completely verified; and (6) provisions for multiple path sensitization were made.

A major feature of HUB and its underlying circuit modelling method, is that one could complete HUB as an ATPG and use it to generate circuit verification stimulus before the complete circuit has been implemented. This is an area of research that has been neglected by most researchers although it is very important to those who need to verify a circuits completeness from a functional aspect. Instead of generating tests for structural faults, one could use the intrinsic ability of binary decision diagrams to create function tests. These vectors could then be reused once the circuit has been implemented and the implementation details are available for fault grading. The remaining faults could then have tests created for them specifically. This use of HUB could also permit the identification of hard to simulate logic allowing the designer to consider circuit modifications before detailed implementation has begun and which might aid in reducing the number of structure related hard faults.

5.3.6 Hierarchy Provisions.

The gBDD circuit model provides the ability to permit hierarchical circuit descriptions. When coupled with the pdcf personality file concept, it is felt that one should be able to work with hierarchically defined circuits and use (re use) precomputed test stimulus for circuit elements described at higher abstraction levels: i.e. and ALU cell instead of its gate level equivalent. This could provide future researchers with the ability to examine the time complexity of ATPG using data base look up methods for large portions of circuitry while using the gBDD to aid in the error propagation and net justification phases. The inherent ability of this hybrid

model to support hierarchical descriptions should permit the user to work at the highest level desired while retaining the ability to descend to the lowest abstraction permitted by the design environment's resources.

5.3.7 Summary.

The algorithm HUB, its use of cost based ATPG constraints and its various features and attributes were explained. The decision to use an automatic test generation based algorithm to identify hard faults was based on the concepts that an algorithm is capable of detecting redundancies and that the test generation would be directly coupled to the measure of test difficulty. FAN, an efficient and extensible algorithm, was choose as the starting algorithm and uses a new circuit modelling technique gBDD. The use of cost based metrics provides a means to link the real cost constraints to test generation and to provide data on which ATPG phases are having difficulty in test generation; an algorithm is also not a decoupled heuristically based approach. Information is fed back to HUB's user permitting the most appropriate corrective actions, based upon the cost constraints imposed by device yield and performance parameters.

6.0 Results.

Several types of results will be examined in this thesis. Although the major thrust was to have been the detection of hard to detect faults within digital combinational logic circuits, it has been expaned to the following 3 areas of research. (1) Some measure that reflect upon the efficiency of the gBDD circuit modelling method; (2) the charactersitics of HUB's ATPG, although not HUB does not contain a complete implementation of FAN; and (3) the verficiation of the hard fault detection process. It is necessary to detail some of the problems encountered during the measurement process in order to forewarn the reader.

Some of the comparisons have been made against a copy of the PODEM ATPG program that was developed by [Ivanov 85] (and which was developed from a version that originated from Hideo Fujiwara's work during his stay at the University of McGill). Unfortunately, there is a minimum of documentation available for this program and its performance in a SUN workstation environment for the larger ISCAS combinational circuits resulted in run time errors (known as "CORE DUMPs"). Thus some comparisons will be limited in their scope due to this.

The ISCAS circuits are described by their neutral netlist; there does not appear to be readily available complete schematic sets. It is this author's belief that for hard fault detection to be truly effective will require that the design's documentation be available (as is currently the case in industry when testability analysis or test pattern generation would be attempted, and the effect of circuit modifications on performance versus objective specifications could be subjectively compared). Although [Marlett 89] suggests that circuit modifications can be performed without the circuit's schematics, this author's beliefs are that hard fault detection and their removal for specific circuits is simply one step in the design process; the long term goal is (should be) to understand the underlying weakness in the design methodology to avoid or eliminate the problem as a natural part of the design methodology. Thus the demonstration of HUB's current ability to detect hard faults will be limited to a small class of circuits for which schematics existed.

Additional measurements will be shown and analysed however.

The ATPG ability of HUB is used as a means to couple the hard fault definition metric to the test generation process (as [Bell Taylor 88] suggest is required) and was not meant to be the development of a new test pattern generator. The programming environment was chosen to ensure that (1) robust code was generated, (2) the primary premise of research could be verified and (3) that a minimum of programming should be performed since this was not an exercise in developing computer science skills. It was assumed that the FAN based ATPG algorithm's time performance would not be seriously degraded by the implementation issues and that, while FAN had been reported to provide a 5 to 6 times improvement over PODEM, small loses of linear improvements in an NP complete problem would not be a significant problem for the purposes of this research.

One final implementation problem was discovered during the final stages of the HUB's development and measurement phases. The original version was started on an IBM PC AT which allowed access to a precise hardware clock contained in the system and which showed the ability to repeat time difference measurements with a high level of confidence. The final version of HUB was developed in the SUN workstation environment to facilitate its creation. However, the UNIX™ operation system does not provide a precise and reproducible time difference with the systems calls that were available. (This was also noticed with a commercial logic simulator that for the same circuit, the same stimulus, and when run on the SUNs provided a factor of 2 for the different run times that it calculated internally.) This appears to be an intrinsic weakness of the UNIX™ operating system which has not appeared to be subjected to the usual cost accounting requirements of an MIS (management of information systems) environment where computer users are charged for their computer resource usage.

6.1 Circuit characteristics of used for results.

Table 6.1.1 provides a summary of the circuit characteristics for the set of combinational circuits that were involved in the measurement process. While not all the circuits were used for each measurement, all of the circuits listed are used at least once.

6.2 Comparison of gBDD.

The program *gBDD* that is used to create the gBDD circuit model is called once per unique circuit net list; this was done to save needless calculations and effort every time that HUB would be used and required only that HUB read in the appropriate circuit data directly into the memory. The translation of the TEGAS like neutral netlist by which the ISCAS and other circuits were described consists of 6 phases: (1) open the original file description, parse the file while verifying the circuit's connectivity, and indicate any errors detected; (2) create a supervisory structure containing centralized data about circuit function and structure; (3) create the graph structure portion of the gBDD circuit model to reflect the circuit's structure; (4) attach the mBDDs for every non UUT_PI and non UUT_PO (currently limited to one output logic elements); (5) calculate the controllabilities and observabilities based upon the COP testability measures and sort the mBDD nodes and fanout branches based upon these values - the fanout stem's observability is the mean of the its branches; and (6) store this data into a file. In this section, the creation of the gBDD's circuit model will be analysed from a generation view point (its time behaviour), and from memory considerations.

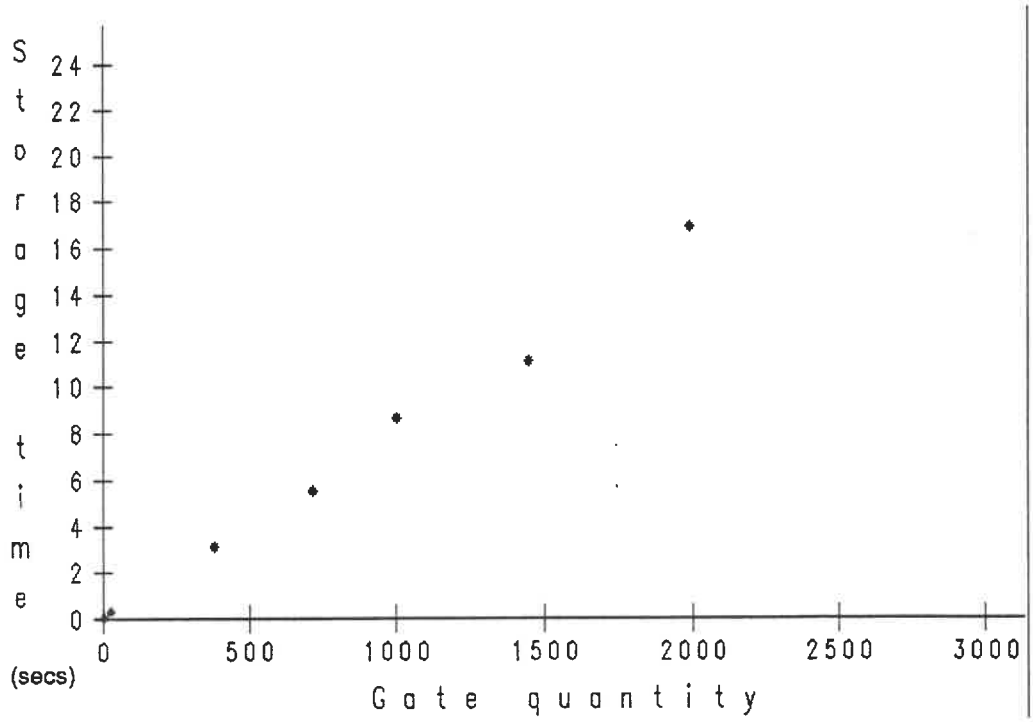
6.2.1 gBDD Temporal performance.

The algorithms used to create the gBDD circuit model from the neutral netlist are essentially linear in performance characteristics. The following graph summarizes the performance of the gBDD algorithms. Once again it is important to remind the reader that the time measurements do not show a high degree of consistency from use to use.

Circuit Name	Gates	Nets	PIs	POs
C17	6	17	5	2
C95	27	95	5	7
C880	383	880	60	26
C1908	718	1908	33	25
C2670	1003	2670	157	54
C3540	1446	3540	50	22
C5315	1994	5315	178	117
C6288	2416	6288	32	32
C7552	2980	7552	206	102
IMAGE	31	80	10	1
ADDER	22	80	3	2

Table 6.1.1: ISCAS and other test circuit characteristics.

The performance for the file storage and reading in the file is linear in the number of gates, equivalent to being linear in the number of graph nodes in the gBDD circuit model, which is a reflection of the one pass required to store and read the circuit elements.



Graph 6.2.1.1: gBDD file storage time.

Table 6.2.1.1 summarizes the time spent in the major phase of the gBDD circuit model generation program and for the *restore()* function used by HUB to reconstruct the circuit in memory from the ASCII gBDD file. The parse phase is seen to consume the largest portion of the generation time: it is during this phase that all the network checks, hash table construction, and the majority of the file input operations occur.

Circuit Name	Time in seconds (s)					
	Parse	gBDD()	mBDD()	OBS()	Store	Restore
C17	0.07	0.02	0.02	0.00	0.07	0.20
C95	0.23	0.10	0.02	0.03	0.32	1.00
C880	4.00	1.63	0.38	0.30	3.13	15.50
C1908	10.05	3.75	0.80	0.57	5.52	33.40
C2670	16.83	6.30	1.12	0.80	8.68	48.80
C3540	32.13	7.15	2.22	1.15	11.12	83.30
C5315	113.55	23.43	2.55	1.88	16.92	237.20
C6288	59.18	28.47	5.88	1.98	20.12	165.00
C7552	132.70	28.27	3.65	2.33	24.48	291.60

Table 6.2.1.1: gBDD time distributions.

6.2.2 File sizes.

HUB uses an ASCII input file format that was selected for ease of development and not based upon efficient or optimal file structures as would be the case for a commercial product. The files' size are compared against 2 different file formats used by PODEM: the first file format, PODEM[1], is an ASCII file format; the second file format is unspecified but appears to be a compressed format for use by PODEM only.

Circuit Name	gBDD (kb)	PODEM[1] (kb)	PODEM[2] (kb)
C17	1.9	2.2	1.1
C95	7.7	9.9	5.2
C880	88.9	88.0	49.0
C1908	152.8	151.7	85.9
C2670	229.5	218.0	125.2
C3540	307.6	293.9	167.7
C5315	469.4	456.8	266.8
C6288	535.7	627.5	351.9
C7552	661.9	631.2	366.0
IMAGE	7.8	8.3	4.1
ADDER	2.5	3.0	1.5

PODEM[1]- ASCII Format.
PODEM[2] - Unspecified Format.

Table 6.2.2.1: Circuit Model File Requirements (k bytes).

Basically PODEM[1] and the gBDD file size are of comparable sizes and are of the same order of magnitude. There is approximately a factor of 2 between the PODEM[2] and the gBDD file sizes although, once again, they are of the same order of magnitude.

6.3 ATPG related results.

HUB will perform its own fault simulation, if requested upon invocation, and operates in two basic modes whether or not fault simulation is used: (1) with fault reduction or (2) without fault reduction. As a basic verification of HUB's instrinsic automatic test pattern generation, its performance against PODEM (without using fault simulation and without fault reduction) was verified for four of the smaller circuits. These four were selected due to the available schematics and since two of the four circuits have known and readily identifiable redundancies (C95 and Image) included. A secondary reason for selecting these four was due to reduced amount of UUT_PIs; HUB does not attempt to maximize the number of these primary inputs which can be set during the generation of a test vector while PODEM appears to provide dynamic vector compression by maximizing the number of the unit under test's primary inputs that can be set without causing a conflict for the target fault.

Circuit	Vector Quantity		Total Time (sec)		Fault Coverage (%)	
	PODEM*	HUB	PODEM*	HUB	PODEM*	HUB
C17	38	15	0.6	0.3	100	100
C95	199	63	6.2	4.3	97.5	97.5
Adder	56	30	1.0	0.8	100	100
Image	149	64	7.1	4.9	92.0	92.0

Table 6.3.1: Partial ATPG results - no simulation.

Table 6.3.1 summarizes the results of this basic comparison. The fault coverage and the execution times are comparable while the quantity of vectors generated tends to be different. This is due to a fundamental difference in the way that HUB works; HUB generates a vector per *pdcf*, from the pdcf personality file described in Chapter 4, as opposed to PODEM which generated a vector per targeted fault when no fault simulation is used. Since the pdcf personality files contain far fewer pdcfs than PODEM contains faults, due to the localized fault quantity reduction used in building the pdcf personality files, HUB will attempt to build less test vectors.

As previously stated, HUB does not attempt to perform any form of dynamic vector compression during automatic test pattern generation for hard fault detection. The amount of logic affected by the UUT_PIs define a *cone of influence* and this will be circuit dependent. Since HUB's primary objective is to detect the presenece of hard faults and not simply provide test pattern generation, an explicit objective of the alogorithm was to set and justifiy only the nodes (and hence eventually only the UUT_PIs) necessary. When the number of UUT_PIs is large, but the number of these inputs which are involved in the cone of influence for which HUB is evaluating, HUB can have an undesired side effect. HUB is expected to generate more vectors than PODEM, due to the lack of this feature as the ratio of the number of UUT_PIs in the cone of influence for the current targeted fault to the total number of UUT_PIs decreases.

Although no attempt was made to determine a heuristic for this test vector reduction effect, the preceding four circuits see a reduction of roughly 2 to 3 times.

6.4 Fault Simulation related results.

The same four circuits were retried but using HUB's fault simulation mode with the fault reduction option activated. For comparison purposes, a commercial logic and fault simulator *SILOS II*™ was used to perform the fault simulation. SILOS II™'s data net list requirements and input pattern requirements required that a small translator (a simple linear algorithm) be built and that the test vectors generated be

time stamped (have a time value associated with it). It was discovered, using the circuits in Table 6.4.1, that where 100% fault detection was achieved with HUB, that SILOS II™ concurred. For the other cases, SILOS II™ provides different fault coverage than HUB determines. This was also found to be the case with larger circuits, notably the C880 and C1908 circuits, and appears to be due to the fault simulation/fault reduction methodology used by HUB. The cause is felt to be implementation based and non critical to the underlying research.

Circuit	Vector Quantity		Total Time (s)		Fault Coverage (%)	
	HUB	FSIM	HUB	FSIM	HUB	FSIM
C17	10	10	0.6	0.4	100	100
C95	20	20	2.5	4.5	91	97
Adder	8	8	0.7	0.6	100	100
Image	48	48	5.5	4.3	91	94

Table 6.4.1: ATPG results with simulation *vs* fault simulation.

It should also be noted that PODEM and SILOS II™ did not always agree with their fault coverage determination, although PODEM was much closer to SILOS II's™ values. SILOS II's™ fault reduction and fault simulation algorithms are not documented and are not available for comparison purposes.

HUB, PODEM, and SILOS II™ did not agree on the total potential fault quantities present for either the reduced (not shown by PODEM) or the total fault count. SILOS II™ provides for some fault reduction, removal of *redundant faults*; no precise definition is provided although they are apparently based upon topological considerations. SILOS II™ faults every net, at input and output pins, as *stuck at 0*

and *stuck at 1*; PODEM also faults every net with these same faults. The differences appear to be part of a more global standardization problem.

6.5 Hard Fault Detection.

Four circuits were examined in detail given the readily availability of their schematics and the relatively small size which permitted to determine whether HUB provides a useful function in identifying the presence of HFs (and redundancies) and presenting information about the reasons that these faults exist.

The underlying premise of this research is the use of cost based constraints coupled with an efficient algorithmic automatic test pattern generation tool to detect the presence of hard faults, as defined by the cost constraints, and the presence of redundant faults as identified by exhaustive searches (as opposed to a heuristic approach such as was used in VICTOR). To avoid the problems created by the lack of reproducibility in the timing of HUB's major ATPG phases when it would be invoked several times for the same circuit but with different cost constraint values, HUB was used with a backtrack limit of 10, previously suggested as a reasonable limit for both PODEM and FAN, but with an unlimited time budget. The data for the circuits was logged and then subsequently it could be manipulated by the RS/1™ tool in order to see the effect of the tighter cost constraints.

Specifically, the graph of the total costs will provide information about how the costs are distributed for a given circuit allowing the ability to compare the effect of tighter and relaxed cost budgets.

A more detailed analysis is provided for a smaller circuit, Image, to demonstrate HUB's instrinsic capabilities.

6.5.1 Extended example using Image.

A portion of the Image circuit was used to examine whether HUB would detect the presence of the hard to test faults using the single stuck at fault model based upon the *pdcf personality file* fault dictionary and provide accurate information

about their causes. This circuit contains untestable faults caused by intentional design redundancies. There are 3 sections to the results for this circuit: (1) the cost accounting information in histogram form showing how the costs for the major automatic test generation phases are distributed plus the budget distributions on a per pdcf basis for HUB without fault simulation and without fault reduction; (2) histograms for the primary and secondary backtrack phases (the phases which caused the backtrack to occur) and for the graph nodes instigating the backtrack; and (3) an indication of the data provided by HUB as to what was occurring when the backtrack situation developed - what heuristics and rules were involved. Please note that the frequency for cost/budgets is in terms of pdcfs; for the backtrack histograms, frequency is in terms of backtrack quantity.

Figure 6.5.1.1 is the circuit representation of the Image circuit. The nets are identified as per the names given in the neutral netlist circuit description. Appendix B contains a cross reference of the graph node to the logic elements' output net name.

Graph 6.5.1.1 summarizes the primary ATPG phase in which HUB was involved and that caused a backtrack to occur. Both the primary and secondary phases (Graph 6.5.1.2) were found to occur principally due to net justification phases. The possibility of difference between the two phases was allowed since the major phase might be net justification but the conflict occurs during the implication phase of the net justification and not due only to net justification. For Image, the propagation phase was also responsible for generating backtracks but these backtrack represent only 25% of the total backtracks which occurred. It is important to remember that HUB keeps track of total unique backtracks and the totals for all the phases. The total backtrack quantity is not simply the addition of the individual phases' backtracks. It was found that the majority of the backtracks occurred in the justification phase.

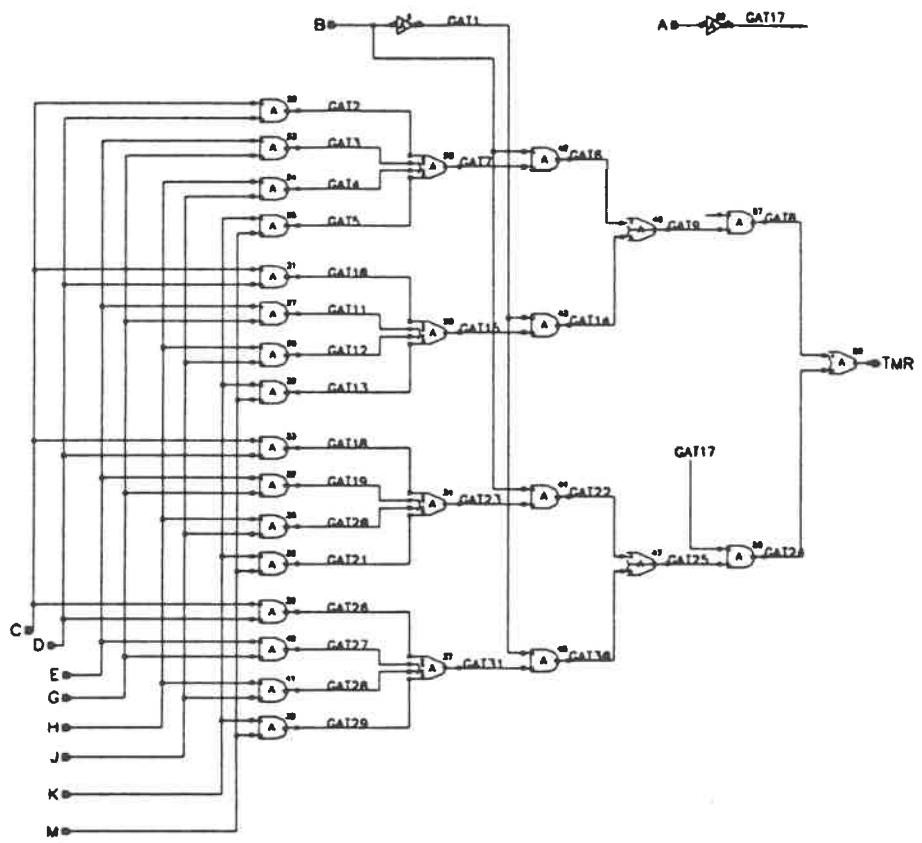
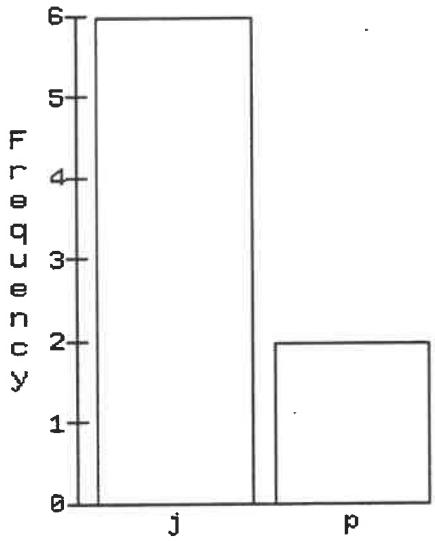
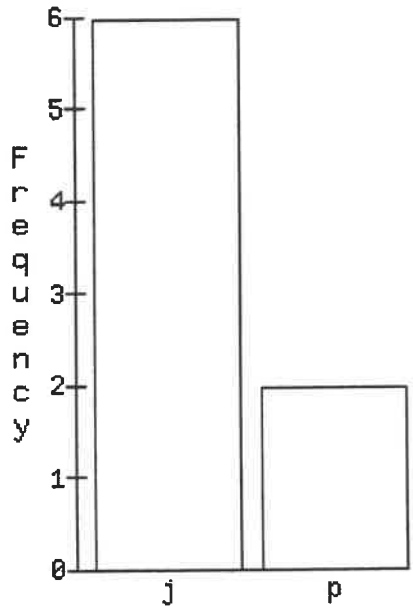


Figure 6.5.1.1: Image circuit subsection



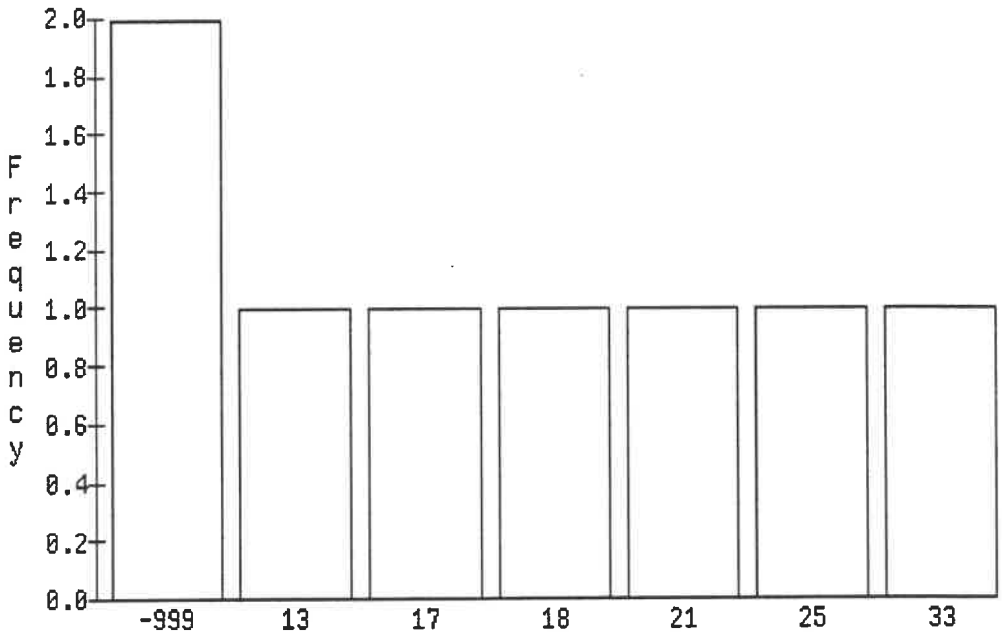
Phases:
p - propagation
j - justification
i - implication
s - sensitization

Graph 6.5.1.1: Primary Backtrack Phases.



Phases:
p - propagation
j - justification
i - implication
s - sensitization

Graph 6.5.1.2: Secondary Backtrack Phases.



Graph 6.5.1.3: Graph Node causing Backtrack Phase.

The histogram in Graph 6.5.1.3 summarizes the information about which graph nodes caused backtracks to occur when this could be uniquely determined and based upon the following simple rules:

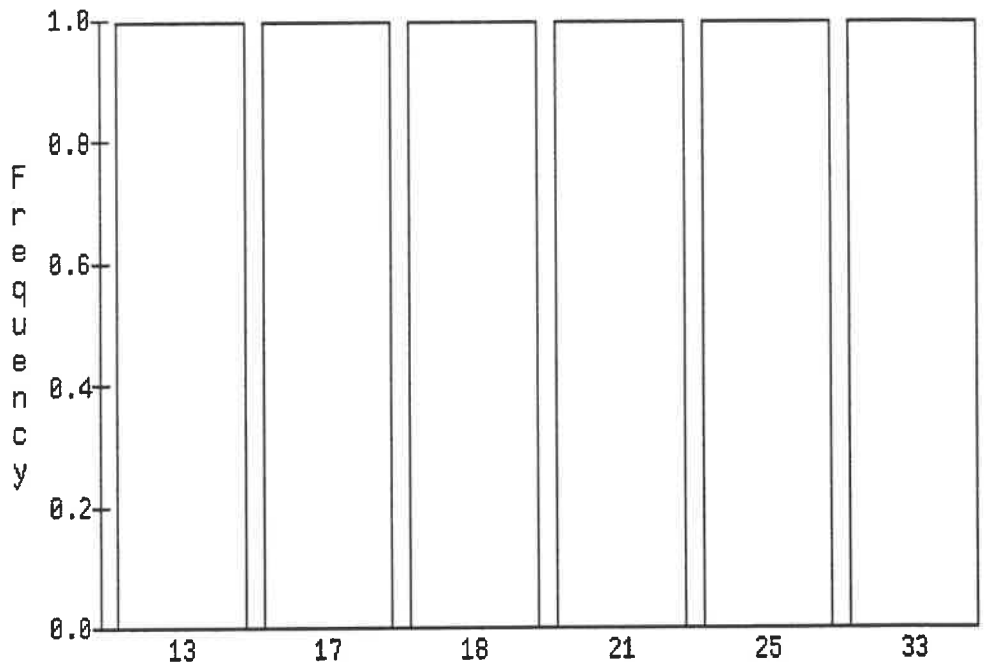
- the graph node's number that can not be justified;
- the graph node's number that blocks the error propagation during the propagation phase at that node;
- the graph node's number that stops the back trace.

If the graph node's identity is not known, then a default value is set to *-999* to be certain that none of the GNs in the range [0 ... *n*] is incorrectly identified.

When the UUT's fault dictionary was reduced for Image, the unknown graph nodes were eliminated, Graph 6.5.1.4, while retaining the remaining 6 graph nodes of Graph 6.5.1.3.

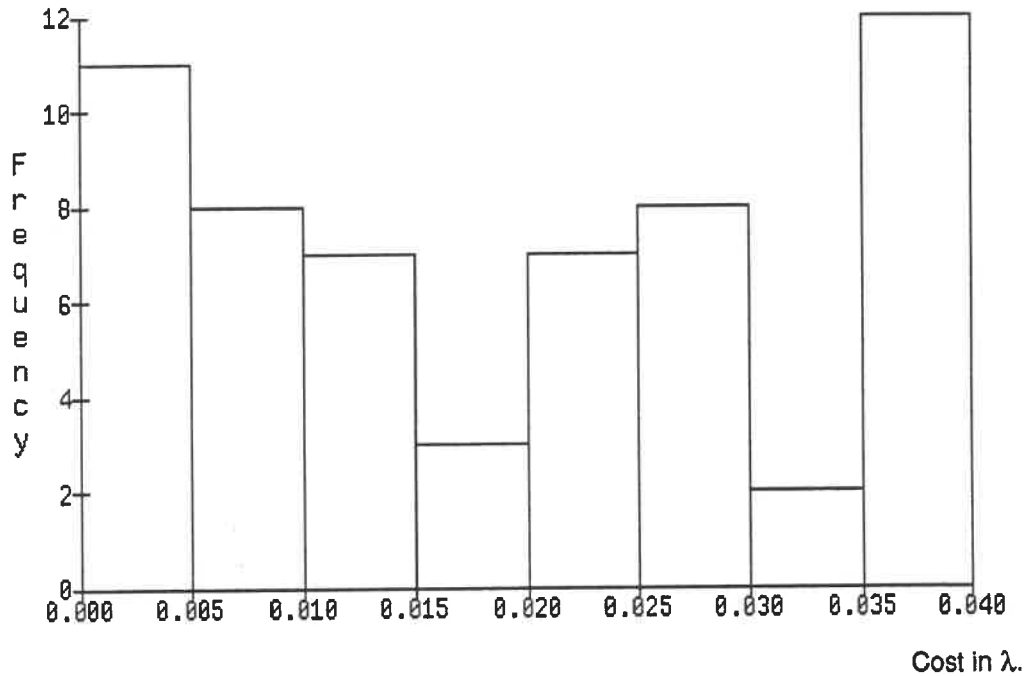
As mentioned, these results (summarizing the various causes of the backtracks encountered) are on a pdcf basis. HUB does not maintain a summary of the backtrack phases (the phases causing a backtrack condition to occur) on a graph node basis as it does for the cost accounting aspects. The author felt that backtracks could possibly occur in different phases for the different pdcfs of a given graph node. Collecting the backtrack causes on a per pdcf basis also allows the association of a fault's identity with the backtrack causes by virtue of the pdcf's data content. This was considered to be an efficient implementation..

Histogram of graph node causing backtracks.

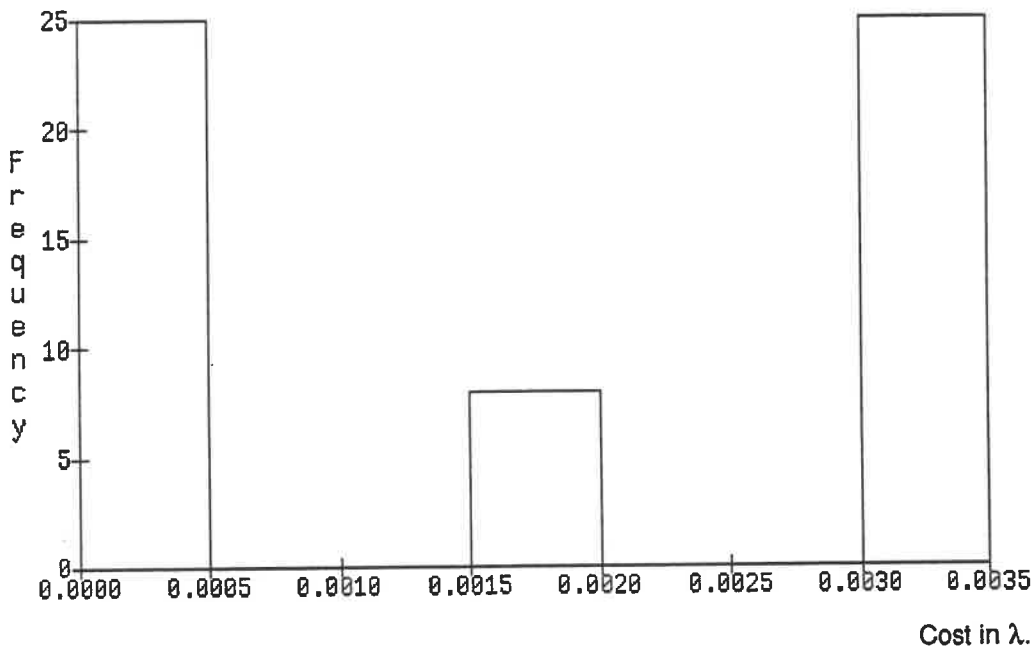


Graph 6.5.1.4: Graph node number (reduced fault dictionary).

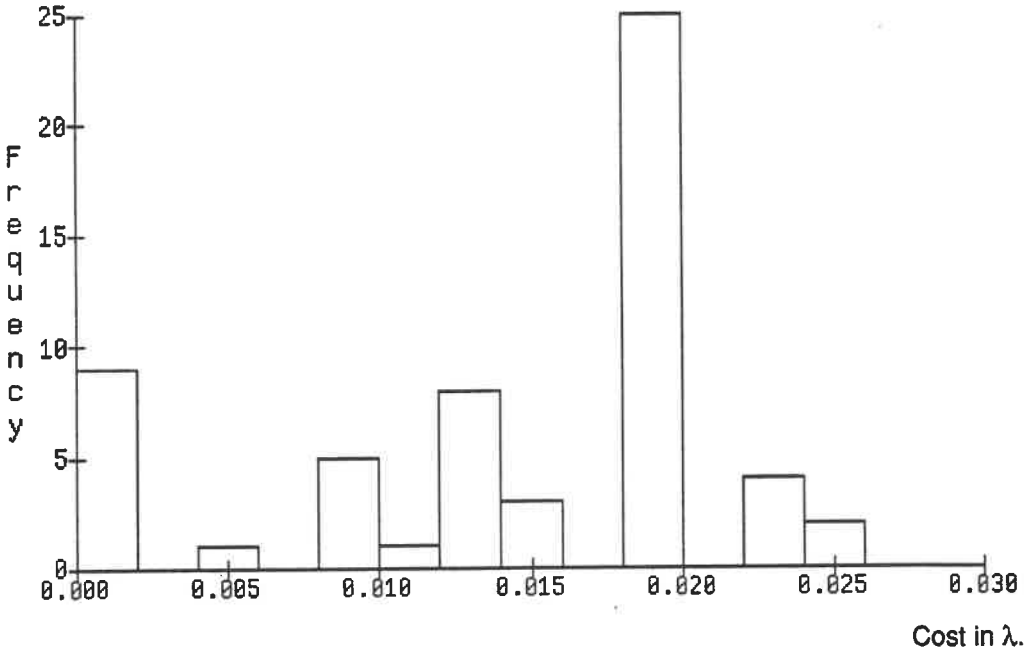
The cost monitoring is done for the propagation, justification, backtrace, backtrack, and implication phases on a pdcf and a graph node basis. This is also the case for the total cost monitoring and for the dynamic budget constraints. The next 8 histograms summarize this information for the Image circuit: the first 7 histograms are for HUB when no fault simulation or reduction was allowed; the eighth histogram is the total cost when HUB's fault simulator, using fault reduction, was used.



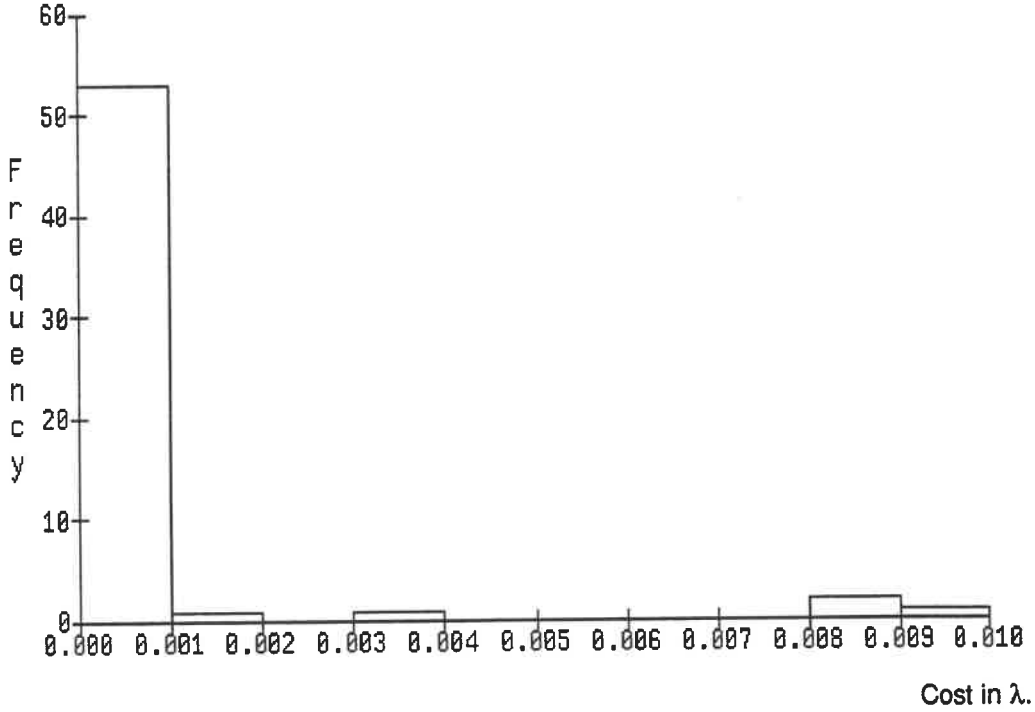
Graph 6.5.1.5: Propagation phase cost histogram.



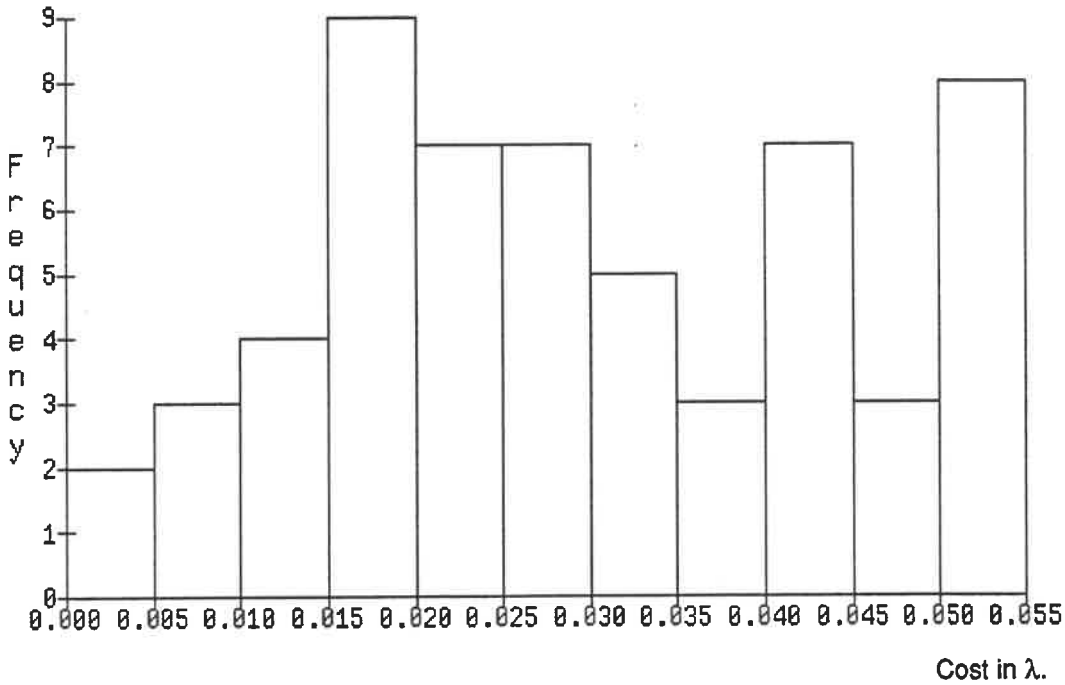
Graph 6.5.1.6: Justification phase cost istogram.



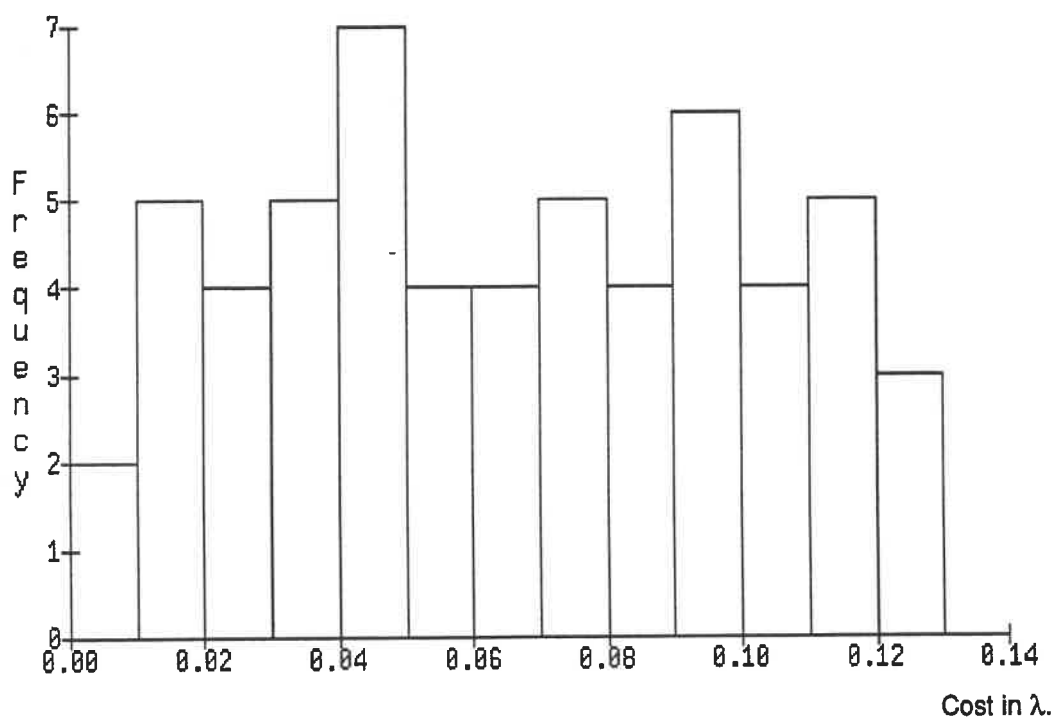
Graph 6.5.1.7: Backtrace phase cost histogram.



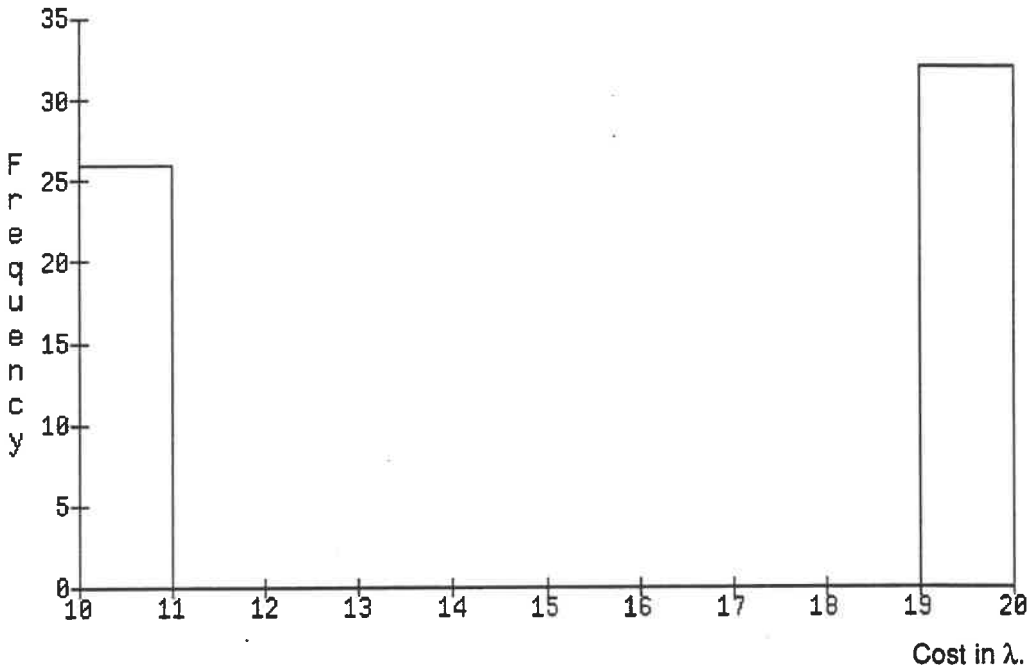
Graph 6.5.1.8: Backtrack phase cost histogram.



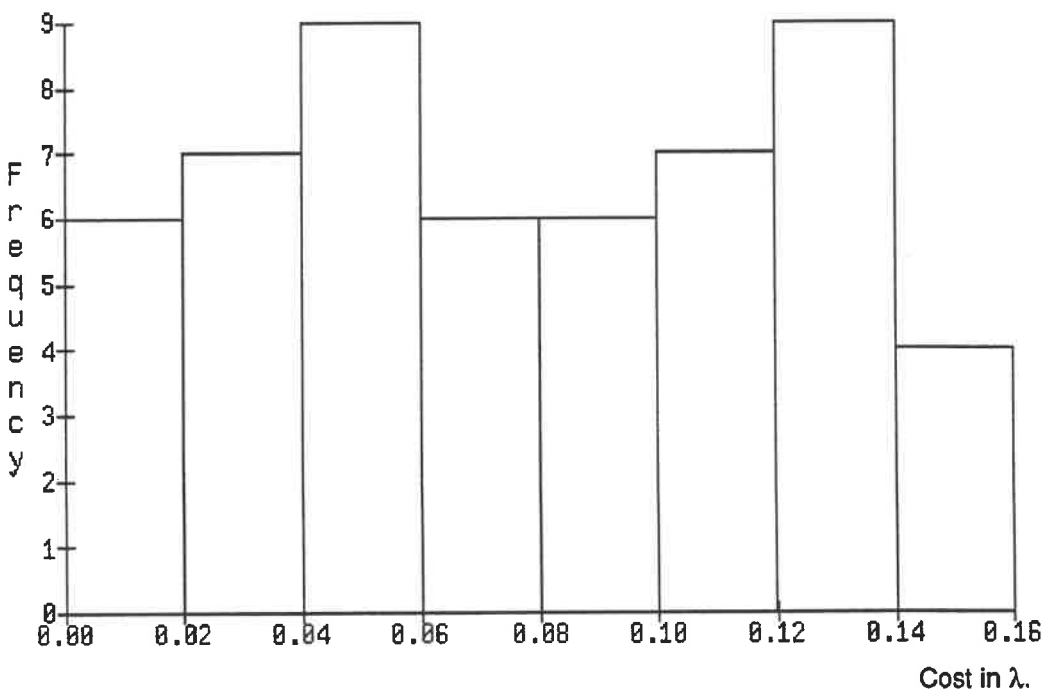
Graph 6.5.1.9: Implication phase cost histogram.



Graph 6.5.1.10: Total cost histogram.



Graph 6.5.1.11: Budget histogram.



Graph 6.5.1.12: Total cost histogram (fault simulation).

The cost λ, is related to the microsecond value returned by the program's call to a UNIX™ system routine. (The absolute value of the time units is 10 of microseconds.) The times are not normalized and are therefore not valid for comparisons between histograms.

The budget reflects that the fault quantity is discrete and, for this circuit, based on either 1 or 2 target faults per pdcf. HUB assigned 10 time units per fault, resulting in the value of 10 and 20 time units. Note that while for this circuit the budgeted time is an integer, this need not be the case when there is a large number of faults. The cumulative frequency of this histogram is the total potential faults for the UUT when no fault simulation is used. The remaining histograms should, and do show a more continuous spectrum of values which mirror the relative degrees of difficulty in each ATPG phase. For Image, it the propagation and implication phases are the most costly test generation phases that were invoked.

Looking at the cumulative cost accounting data (for the propagation and implication phases) and for the total of all the major ATPG phases, one can see the effect as to how many pdcfs would be aborted and result in *potential HFs* if the budget constraints were modified from their unlimited (but respecting a maximum of 10 backtracks) values. This per pdcf data includes the pdcfs for which no test could be completed due to the presence of redundancies. Thus if the budget constraints are sufficiently severe, even redundant faults would be classified as potential hard faults and become hard faults if not detected by test stimulus which respected the more stringent cost constraints. Table 6.5.1.1 details the cumulative costs and the number of pdcfs which respect the various thresholds for the total of the circuit costs. Tables 6.5.1.2 and 6.5.1.3 provide the individual information for the propagation and justification phases respectively.

Time Threshold (time units * 1000)	Quantity of pdcf respecting Time Threshold
130	58
120	55
110	50
100	46
90	40
80	36
70	31
60	27
50	23
40	16
30	11
20	7
10	2

Table 6.5.1.1: Cumulative total cost distribution.

Time Threshold (λ * 1000)	Quantity of pdcf respecting Time Threshold
40	58
35	46
30	44
25	36
20	29
15	26
10	19
5	11

Table 6.5.1.2: Cumulative propagation distribution.

Time Threshold (λ * 1000)	Quantity of pdcf respecting Time Threshold
55	58
50	55
45	50
40	46
35	40
30	36
25	31
20	27
15	23
10	16
5	11

Table 6.5.1.3: Cumulative justification cost distribution.

The lower bound of 1 fault per pdcf is 1 and the upper bound is 2 faults per pdcf for the circuits used in this research. Thus one can equate this knowledge into lower and upper bounds for the potential hard faults.

Figure 6.5.1.2 is a portion of the information logged by HUB for the Image circuit. An exhaustive search has been performed for graph node 10 (the inverter with output net *GAT17*, from the list in Appendix B), specifically for the inverter pdcf 0 (output *s-a-1*). HUB provides data on the failure to generate a test using this pdcf; in this case the failure is due to the inability to propagate the difference value at the *2-input* OR gate, GN 40 having the output net *TMR*. HUB prints the number of the rule violated and explains this rule: P.8.5 meaning that one of the mBDD's nodes blocked the error propagation phase - *!R. 1 leaf & picked R* indicating that the rule required the *left branch* to be picked but that the previously set value causes the *right leaf* to be picked at a node which has one branch and one leaf. Since for FAN, the search space is a function of the number of *head lines*, HUB will tried to provide a solution by retrying all the possibilities associated with the search space.

It is important to note that HUB will always create data about the costs incurred as it tries to create a test from the pdcf used to sensitize a fault even if no backtracks occur. Thus, in the presence of resistant faults due to non redundancies, the cost information identifies which phase is the most costly: the basis of this approach was that one could then implement a more global improvement in making faults testable by the appropriate and cost effective method (an internal testability bus to allow isolation, improved controllability and observability of the basic circuit building blocks). Hooks have been placed into HUB which would allow visualization of the activated paths back into a design environment when such a tool exists; this would probably be more useful than the simple cost accounting information.

```
RULEP.8.5
Error in propagation of BDD
GNODE  OR2 PO_num 40.  BDD i.d.  1
for NUT  10
!R: 1 leaf and picked R.
RULEP.8.5
Error in propagation of BDD
GNODE  OR2 PO_num 40.  BDD i.d.  1
for NUT  10
!R: 1 leaf and picked R.
RULEP.8.5
Error in propagation of BDD
GNODE  OR2 PO_num 40.  BDD i.d.  1
for NUT  10
!R: 1 leaf and picked R.
RULEP.8.5
Error in propagation of BDD
GNODE  OR2 PO_num 40.  BDD i.d.  1
for NUT  10
!R: 1 leaf and picked R.
Exhaustively  searched!  Redundancy!
(10 : 0) No Success
Determine_Reasons not yet implemented
```

Figure 6.5.1.2: Partial HUB hard fault data.

6.5.2 General Results.

HUB was used on 5 of the 1985 ISCAS combinational circuits (C17, C95, C880, C1908, and C2670) plus the full adder cell and the Image circuit cell. The following tables and histograms summarize the costs, budgets, backtrack information and general data resulting from these measurements.

The measurements show that the biggest cause of backtracks is the net justification phase, and that HUB had equal quantities of justification induced backtracks for the primary and secondary causes. Table 6.5.2.1 summarizes these

results and also includes data on the number of unique graph nodes involved in generating the backtrack condition. No work was undertaken to determine the effect of modifying the graph node to allow easier justification although Marlett has used the concept of *T-cells* to automatically modify the circuit based upon the most frequent element involved in stopping test generation and then allowing the continuation of the ATPG process [Marlett 89].

Circuit Name	Primary[1]	Secondary[1]	GN Quantity[2]
C17	N/A	N/A	N/A
C95	6	6	6
C880	28	28	11
C1908	140	140	12
C2670	580	580	580
IMAGE	250	250	98
ADDER	N/A	N/A	N/A

Notes:

1) The number of justification caused backtracks. These represent the amount of backtracks which occurred in as the result of line justification being either the primary or secondary cause.

2) The quantity of graph nodes which were identifiable as the source of the conflict that originated the backtrack process. A given node can cause more than one backtrack.

Table 6.5.2.1: Justification backtrack information.

In general, for these circuits, it was found that propagation induced backtracks were the only other major backtrack component; implication, backtrace, and backtrack induced backtracks were virtually non existent. Tables 6.5.2.2 and 6.5.2.3 presents the average number of backtracks for each of the circuits tried: the full adder and the C17 circuit are fully testable and did not cause any backtracks: any backtracks would have been due to poor guidance heuristics since there is no topological reasons for

backtracks.

	Backtrack Quantity			
Circuit Name	Total	Justification	Propagation	Implication
C17	N/A	N/A	N/A	N/A
C95	0.558	0.474	0.032	N/A
C880	1.184	0.249	0.002	N/A
C1908	5.916	1.565	0.065	N/A
C2670	2.194	0.464	0.136	N/A
IMAGE	0.931	0.345	0.138	N/A
ADDER	N/A	N/A	N/A	N/A

N/A - not applicable.

Table 6.5.2.2: Mean of backtracks.

Tables 6.5.2.4 through 6.5.2.8 summarize the means for the cost of the various ATPG phases in HUB for the 7 circuits evaluated. It is interesting to note that all but 2 of the costs do not exhibit a monotonically increasing cost with complexity function. However, the error propagation and implication phases do seem to possess this monotonicity. *The time or cost is in λ units for this thesis.*

Circuit Name	Mean (λ)	Standard Deviation
C17	10 000	6 172
C95	85 891	47 551
C880	2 266 207	1 609 804
C1908	16 393 475	12 282 399
C2670	10 155 143	7 338 218
IMAGE	66 808	35 482
ADDER	4 583	3 622

Table 6.5.2.3: Mean and standard deviation of total costs.

The standard deviation in many cases is as large if not larger than the mean. This is not surprising considering the data distributions of these cost components: the histograms for the total ATPG costs of the seven circuit examined using HUB are found in Appendix C.

Circuit Name	Mean (λ)	Standard Deviation
C17	3 444	3 241
C95	9 193	4 650
C880	291 018	177 310
C1908	418 582	648 307
C2670	518 490	333 983
IMAGE	19 913	12 080
ADDER	Not Defined	Not Defined

Table 6.5.2.4: Mean and standard deviation of propagation total costs.

Circuit Name	Mean (λ)	Standard Deviation
C17	Not Defined	Not Defined
C95	13 298	8 258
C880	97 866	68 549
C1908	42 994	12 403
C2670	516 629	281 879
IMAGE	1 667	1 561
ADDER	Not Defined	Not Defined

Table 6.5.2.5: Mean and standard deviation of justification total costs.

Circuit Name	Mean (λ)	Standard Deviation
C17	Not Defined	Not Defined
C95	15 192	15 549
C880	81 728	58 130
C1908	7 044 623	7 582 245
C2670	886 330	789 318
IMAGE	546	2 032
ADDER	Not Defined	Not Defined

Table 6.5.2.6: Mean and standard deviation of backtrack total costs.

Circuit Name	Mean (λ)	Standard Deviation
C17	6 555	3 301
C95	24 788	13 358
C880	660 862	477 866
C1908	4 465 937	2 714 843
C2670	2 887 644	2 114 447
IMAGE	14 166	7 100
ADDER	3 083	1 733

Table 6.5.2.7: Mean and standard deviation of backtrace total costs.

Circuit Name	Mean (λ)	Standard Deviation
C17	Not Defined	Not Defined
C95	23 420	7 998
C880	1 134 732	834 495
C1908	4 421 338	2 261 968
C2670	5 346 050	3 851 943
IMAGE	30 516	14 746
ADDER	1 083	1 555

Table 6.5.2.8: Mean and standard deviation of implication total costs.

Although HUB is able to generate the cost information as a means to identify hard to test faults, even when there are no backtracks and the fault is non redundant, it will be necessary to produce additional information which would indicate how circuit modifications could be implemented and cause the fault to become easily testable. One possible method would be to indicate on the circuit's schematics, at the high abstraction level possible, which circuit macros and nets have been activated and manipulated; this would allow human intervention. This is a major weakness to HUB at present.

From an implementational aspect, HUB requires many extra tools to provide manipulation of the large amounts of data that HUB is capable of producing; BBN's RS/1™ statistical package was used - a tool that would be non trival to construct during the process of a M.Sc.A.. Even with this tool, HUB may produce a surplus of data which could be pruned into a more useful quantity.

6.6 Summary.

A variety of measurements were made using the HUB environment; the

gBDD circuit model, the characteristics of HUB's automatic test pattern generator and its fault simulator, and to indicate HUB's ability to produce detailed information about the existence and causes of hard to test faults with a combinational logic circuit. The results were obtained by using a subset of the 1985 ISCAS combinational logic circuits and two additional circuits: a full adder cell and a portion of the Image circuit, a circuit developed at École Polytechnique.

HUB succeeds in providing useful information for the design team and in coupling the concept of testability measurement to the ATPG process, but additional information and efforts will be required before the detection of hard to test faults is completely solved.

7.0 Conclusions.

An attempt to identify the presence of hard to test faults in a digital combinational circuit by a new method was tried The underlying concepts of the this method involved the idea of coupling cost constraints with an efficient automatic pattern generator (ATPG). The cost constraints are in the form of budgets and they can be compared dynamically with the ATPG costs; the costs are obtained by monitoring the important phases of an ATPG phase that is based upon the FAN algorithm. A secondary set of concepts consists of identifying which ATPG phases cause backtracks to occur, and the quantity of each, plus determining the logic element which is clearly responsable in initiating the backtrack condition. A computer implementation, HUB (*hard fault detection using budget constraints*), was used to evaluate the basic principles of these concepts.

HUB consists of a modified version of the FAN ATPG algorithm which allows the monitoring of the ATPG phases' costs. HUB uses traditional testability measures (COP) to provide partial aid in selecting nodes during the error propagation and net justification phases, in addition to error propagation heuristics included in a new circuit modelling technique created to aid HUB's automatic test generation process. An environment for HUB's use was also created and consits of: (1) a circuit netlist translator (from a neutral netlist format) to the new gBDD (*graph binary decision diagram*) circuit model used by HUB; (2) a simple single stuck at fault simulator; (3) translation tools that permit HUB's test vectors and gBDD's circuit description to be converted into the input format required by a commercial logic and fault simulator (SIMUCAD's SILOS II™); and (4) output data files, from HUB, in an ASCII format which were used by BBN's RS/1™ statistical analysis package in providing meaningful data.

Some measurements were effected using circuits from the 1985 ISCAS combinational benchmark circuits plus two additional circuits. The measurements centered upon 3 arenas: the gBDD circuit model's characteristics; HUB's ATPG and fault simulation characteristics; and HUB's hard fault detection abilities.

The gBDD circuit modelling method would appear to be the single most useful result of this research albeit that the thesis' thrust was to find a method to detect HFs based upon cost considerations. The gBDD circuit model has provided a method to combine a circuit's structural and functional information in a reasonably compact format which did not cause performance problems for the ATPG process. This model reinforces the positive aspects of the previously uncombined graph model and the binary decision diagram circuit (BDD) descriptions which each have major handicaps: graph modelling do not provide any functional details while the binary decision diagrams preclude structural knowledge at the cost of poor single stuck at fault coverage.

The resulting gBDD model permits the ability to describe a circuit in a hierarchical manner, to generate tests based upon non traditional fault models (that is a functionality based fault model referred to as an *experiment*), and to incorporate previous research based upon graph models and BDDs. This model was invented expressly to aid the test pattern generation process and allow the future expansion of the HUB tool to mixed sequential combinational logic circuits. Further advantages of the modelling method will allow the construction of test stimulus before the full implementation details of the circuit are known - in particular shortly after the basic circuit's building blocks and the interconnecting data paths are known - so that efforts at detecting hard faults, from a functional testing aspect, can be identified before the design has proceeded a great length.

HUB's ATPG capabilities, although not a complete implementation of Fujiwara's FAN algorithm and without dynamic vector compaction, has performance characteristics similar to those of a university copy of PODEM. Due to the lack of dynamic vector compression, HUB will tend to generate more test vectors for circuits having large numbers of primary inputs, but few primary inputs per cone of influence.

HUB's hard fault detection mechanisms were seen to be functional, although they may not be sufficiently useful when no backtracks occur. If one can reduce the amount of backtracks by efficient algorithms and valid heuristics, then HUB's cost

accounting information will need to be augmented by additional data to provide useful feedback to the design team.

HUB was seen to generate correct test vectors for circuits and its information did provide accurate data about the cause of hard to test faults for the circuits for which schematics could be used to verify HUB's results. The cost information did not show that one particular phase was generally responsible for the majority of resource usage although, for all the circuits evaluated, it was seen that HUB clearly identified that the majority of backtracks occurring were initiated by the net justification phase.

There are several next steps which HUB and the gBDD circuit model could naturally be extended to encompass. The circuit model should allow the design team to evaluate testability aspects without having the complete implementation specified due to its functional modelling aspects.

HUB has other abilities as yet unexplored and it would be desirable to incorporate this tool into an integrated design evironment such as the CADENCE company has attempted to do by integrating electronic schematic capture environment with links to logic simulators and analog simulation tools from within the one tool. Adding a testability analyzer would be of use especially in a paradigm that supports hierarchical circuit descriptions. (Note that CADENCE supports SCOAP whose limitations as a testability metric were previously described.)

8.0 Bibliography.

[Abadir Reghbati 85] Abadir, M.S. and Reghbati, H. K.,
"*Functional Test Generation for LSI Circuits Described by Binary Decision Diagrams,*" Proceedings of Intl. Test Conf., 1985, pp. 483 - 492.

[Abadir Reghbati 86] Abadir, M.S. and Reghbati, H. K.,
"*Functional Test Generation for Digital Circuits Described by Binary Decision Diagrams,*" IEEE Trans. on Comp., Vol. C-35, 1985, pp. 375 - 379.

[Abramovici 89] Abramovici, M., and Miller, D.T.,
"*Are Random Vectors Useful in Test Generation?,*"
Proceedings of European Test Conf. 1989, pp. 22 - 25.

[Agrawal Mercer 82] Agrawal, V.D. and Mercer, M.R.,
"*Testability Measures - What Do They Tell Us?,*"
Proceedings of Intl. Test Conf., 1982, pp. 391 - 396

[Agrawal Seth. 89] Agrawal, V.D., and Seth., S.,
"*Tutorial : Test Generation for VLSI Chips,*"
IEEE Computer Society Press, Washington, 1988.

[Agrawal et al. 88] Agrawal, V.D., Cheng, S., Johnson, D.D.,
"*Designing Circuits with Partial Scan,*"
IEEE Design and Test of Comp., Vol. 5, No. 2, 1988, pp. 8 - 15.

[Agrawal et al. 89] Agrawal, V.D., Kwang-Ting, C., and Agrawal, P.,
"*A Directed Search Method for Test Generation Using a Concurrent Simulator,*" IEEE Trans. on Comp. Aided Design, Vol. 8, No. 2, Feb 1989, pp. 131 - 138.

[Akers 78a] Akers, S.B.,
"*Binary Decision Diagrams,*"
IEEE Trans. on Comp., Vol. , 1978, pp. 509 - 515.

[Akers 78b] Akers, S.B.,
"*Functional Testing with Binary Decision Diagrams,*"
Proceedings of Fault Tolerant Conf., 1978, pp. 75 - 82.

[Bardell *et al.* 86] Bardell, P.H., McAnney, W.H., and Savir, J.,
"*Built-In Self Test for VLSI: Pseudorandom Techniques,*"
Wiley, Somerset, N.J., 1987.

[Batni Kime 76] Batni, R.P., and Kime, C.R.,
"*A Module-Level TestingApproach for Combinational Networks,*"
IEEE Trans. on Comp., Vol. C-25, No. 6, June 76, pp. 594 - 604.

[Beenker *et al.* 89] Beenker, F., Dekker, R., Stans, R.,
"*A Testability Strategy for Silicon Compilers,*"
Proceedings of Intl. Test Conf., 1989, pp. 660 - 670.

[Bell Taylor 88] Bell, I.M., and Ta;ylor, G.E..,
"*Detection of Reconvergent Fanout Features in Digital Circuits,*"
Proceedings of 1988 Canadian Testability Workshop, pp. 21 - 45

[Brglez 83] Brglez, F.,
"*Testability in VLSI,*"
Proceedings of CCVLSI, 1983.

[Brglez 84] Brglez, F., Pownall, P, and Hum, R.,
"*Application of Testability Analysis: from ATPG to Critical Delay Path Tracing,*" Proceedings of Intl. Test Conf., 1984, pp. 705 - 712.

[Brglez 85] Brglez, F.,
"*A Fast Fault Grader: Analysis and Applications,*"
Proceedings of Intl. Test Conf. 1985, pp. 797 - 800.

[Breuer 83] Breuer, M. A.,
"*Automatic Design For Testability Via Testability Measures,*"
Proceedings of Autotestcon, 1983, pp. 138 - 143.

[Calhoun Brglez 89] Calhoun, J.P., and Brglez, F.,
"*A Framework and Method for Hierarchical Test Generation,*"
Proceedings Of Intl. Test Conf., 1989, pp. 480 - 490.

[Chang *et al.* 86] Chang, H.P., Rogers, W.A., Abraham, J.A.,
"*Structured Functional Level Test Generation Using Binary Decision Diagrams,*" Proceedings of Intl. Test Conf., 1986, pp. 97 - 104.

[Chen Breuer 85] Chen, T. and Brueur, M. A.,
"Automatic Design For Testability Via Testability Measures,"
IEEE Trans. on CAD, Vol. CAD-4, No. 1, Jan. 1985, pp. 3 - 11.

[Devadas *et al.* 88] Devadas, S., Ma, H-K. T., Newton, A.R., and Sangiovanni-Vincentelli, A.,
"Synthesis and Optimization Porcedures for Fully and Easily Testable Sequential Machines," Proceedings of Intl. Test Conf., 1988, pp. 621 - 630.

[D & T 89] D & T Roundtable discussion,
"CAD For System Design: Is It Practical,"
IEEE Design and Test of Comp., Vol. 6, No. 2, April 1989, pp. 46 - 54.

[Eldred 59] Eldred, R.D.,
"Test Routines Based on Symbolic Logic Statements,"
Journal A.C.M., No. 1, 1959, pp 33 - 36.

[Ferguson 88] Ferguson, F.J., and Shen, J.P.,
"Extraction and Simulation of Realistic CMOS Faults using Inductive Fault Analysis," Proceedings of Intl. Test Conf., 1988, pp. 475 - 484.

[Fujiwara 83] Fujiwara, F., and Shimono, T.,
"On the Acceleration of Test Generation Algorithms,"
IEEE Trans. on Comp., Vol. C-32, No. 12, Dec. 1983, pp 1137 - 1144.

[Fujiwara 85] Fujiwara, F.,
"FAN: A Fanout - Oriented Test Pattern Generation Algorithm,"
Proceedings of ISCAS, 1985, pp. 671 - 674.

[Galiay 80] Galiay, J., Crouzet. Y., and Vergniault, M.,
"Physical Versus Logical Faults in MOS LSI Circuits: Impact on their Testability,"
IEEE Trans. on Comp., Vol. C-29, No. 6, June 1978, pp. 527 - 531.

[Goldstein 79] Goldstein, L.H.,
"Controllability/Observability Analysis of Digital Circuits,"
IEEE Trans. Circuits Syst., 1979, pp. 685 - 693

[Goldstein 80] Goldstein, L.H., and Thigpen, E.L.,
"*SCOAP: Sandia Controllability/Observability Analysis Program,*"
Proceedings of Design Auto. Conf., 1980, pp. 190 -196.

[Goel 81a] Goel, P.,
"*An implicit enumeration algorithm to generate tests for combinational circuits,*" IEEE Trans. on Comp., Vol. 30, 1981, pp. 215 -222.

[Goel 81b] Goel, P., and Rosales, B.C.,
"*PODEM-X: An automatic test generation system for VLIS logic structures,*" Proceedings of Design Auto. Conf., 1981, pp. 260 -268.

[Henckels 88a] Henckels, L. P.,
Invited Speaker's Presentation,
Proceedings of Intl. Test Conf., 1988.

[Henckels 88b] Henckels, L. P.,
Private correspondance, 1988.

[Huisman 88] Huissman, L.M.,
"*The Reliability of Approximate Testability Measures,*"
IEEE Design and Test of Comp., Vol. xx, No. 6, 1988, pp. 57 - 67.

[Ivanov 85] Ivanov, A.,
"*Dynamic Testability Measures and their User in ATPG,*"
Master's Thesis, University of McGill, July 1985.

[Kirkland Mercer 88] Kirkland, T., and Mercer, M. R.,
"*Algorithms for Automatic Test Pattern Generation,*"
IEEE Design and Test of Comp., Vol. 5, No. 3, June 1988, pp. 42 - 55.

[Kolman Busby 84] Kolman, B., and Busby, R.C.,
"*Discrete Mathmatical Structures for Computer Science,*"
Prentice-Hall, Toronto, 1st Ed., 1984.

[Lisanke *et al.* 86] Lisanke, R., Brglez, F., deGeus, A., and Gregory, D.,
"*Testablity-Driven Random Pattern Generation,*"
Proceedings of ICCAD, 1986, pp. 144 - 147.

[Lisanke *et al.* 87] Lisanke, R., Brglez, F., deGeus, A., and Gregory, D.,

"Testablity-Driven Random Pattern Generation,"
IEEE Trans. on CAD, Vol. CAD-6, No. 6, Nov. 1987, pp. 1082 - 1087.

[Marlett 78] Marlett, R.,
"EBT: A Comprehensive Test Generation Technique for Highly Sequential Circuits," Proceedings of 15th Design Auto. Conf., 1978, pp. 335 - 339

[Marlett 86] Marlett, R.,
"An effective test generation system for sequentail circuits,"
Proceedings of Design Auto. Conf., 1986, pp. 250 -256.

[Marlett 89] Marlett, R.,
"Sequential ATP and Partial Scan,"
Proceedins of IEEE VLSI Testability Workshop, Atlantic City, 1989, pages are not numbered.

[McClusky 86] McClusky, E.J,
"Logic Design Principles With Emphasis on Testable Semicustom Circuits," Prentice-Hall, Englewood, 1986.

[McClusky 88] McClusky, E.J,
"IC Quality and Test Transparency,"
Proceedings of Intl. Test Conf., 1988, pp. 295 - 301.

[Miczo 86] Miczo, A.,
"Digital Logic Testing and Simulation,"
Harper & Row, New York, 1986.

[Min Rogers 89] Min, H.B., and Rogers, W.A.,
"Search Strategy Switching: An Alternative to Increased Backtracking,"
Proceedings of Intl. Test Conf., 1989, pp. 803 - 811.

[Patel Patel 86] Patel, S., and Patel, J.,
"Effectiveness of Heuristics Measured for Automatic Test Pattern Generation," Proceedings of 23rd ACM IEEE DAC, June 1986, pp. 547 - 552.

[Ratiu 82] Ratiu, I.M.., Sangiovanni-Vincentielli, A., and Pederson, D.O.,
"VICTOR: A Fast VLSI Testability Analysis Program,"

Proceedings of Intl. Test Conf., 1982, pp. 397 - 401.

[Roth 67] Roth, J.P., Bouricius, W.G., and Schneider, P.R.,
"*Programmed algorithms to compute tests to detect and distinguish between failures in logic circuits,*" IEEE Trans. on Elect. Comp., Vol. 12, 1967,
pp. 567 - 580.

[Russell Kime 71] Russell, J.D., and Kime, C.R.,
"*Stuctural Factors in the Fault Diagnosis of Combinational Networks,*"
IEEE Trans. on Comp., Vol. C-20, No. 11, Nov. 71, pp. 1276 - 1285.

[Salzmann *et al.* 88] Salzmann, C., Funcell, M., and Taylor, R.,
"*Design For Test and the Cost of Quality,*"
Proceedings of Intl. Test Conf., 1988, pp. 302 - 307.

[Sangio 88] Sangiovanni-Vincentelli, A.,
"*Optimal Logic Synthesis and Testability: Two Sides fof the Same Coin,*"
Invited Speaker's Presentation, 1988 Int. Test Conf.

[Savri 83] Savir, J.,
"*Good Controllability and Observability Do Not Guarantee Good Testability,*" IEEE Trans. on Comp., Vol. C-32, No. 12, Dec. 83,
pp. 1198 - 1200.

[Seth Agrawal 89] Seth., S., and Agrawal, V.D.,
"*Cutting chip testing costs,*"
IEEE Spectrun, Vol. , No. 4, April 1985, pp 38 - 45.

[Soden *et al.* 89] Soden, J., Treece, R.K., Taylor, M.R., and Hawkins, C.F.,
"*CMOS IC Stuck-Open Fault Electrical Effects and Design Considerations,*" Procedings of Intl. Test Conf., 1989, pp 423 - 430.

[Shen 85] Shen, J.P, Maly, W., and Ferguson, F.J.,
"*Inductive fault Analysis of MOS Integrated Circuits,*"
IEEE Design and Test of Comp., Vol. 2, No. 12, 13 - 26.

[Somenzi *et al.* 85] Somenzi, F., Gai, S., Mezzalama, M., and Prinetto, P.,
"*Testing Strategy and Technique for Macro-based Circuits,*"

IEEE Test of Comp., Vol. 2, pp. 85 - 90.

[Stallings 88] Stallings, W.,
"ISDN: An Introduction,"
MacMillan, 1st Edition, 1989.

[Stannard Kaminska 88a] Stannard, D.H., and Kaminska, B.,
"Detection of Hard Faults in a Combinational Circuit Using Budget Constrains," Proceedings of Intl. Test Conf., 1988, pp. 999.

[Stannard Kaminska 88b] Stannard, D.H., and Kaminska, B.,
"The Use of Cost Baed Testability Measures in Binary Decision Diagrams," Proceedings of CCVLSI., 1988, pp. 108 - 113.

[Stannard Kaminska 89a] Stannard, D.H., and Kaminska, B.,
"A New Automatic Test Pattern Generation Based on Cost Driven Heuristics," Proceedings of IEEE VLSI Testability Workshop, Atlantic City., 1989, pages are not numbered.

[Stannard Kaminska 89b] Stannard, D.H., and Kaminska, B.,
"gBDD: An Extended circuit modeling technique to support ATPG,"
Proceedings of CCVLSI., 1989, pp. 81 - 89.

[Wadsack 78] Wadsack, R.L.,
"Fault Modelliing and Logic Simulation of CMOS and MOS Integrated Circuits," Bell Sys. Tech. Journal, Vol. 57, No. 5, May 78.

[Warshall 62] Warshall, S.,
"A Theorem on Boolean Matrices,"
Journal of the Association of Computing Machinery, 1962, pp. 11 - 12.

[Williams 89a] Williams, T.,
Invited Speaker's Presentation,
Proceedings. of Intl. Test Conf., 1989.

9.0 Appendix A

HUB is invoked using the following command syntax, detailed in Figure A.1, and consisting of three main argument components.

hub [- option list] Input_file_name Accounting_file_name

When the fault simulation option is used, an additional file *faultsdetecter.sim* is created and which contains summarized data about the fault simulation results. The file *input_file.extension* must be in the *gBDD* file format; a simple check is made and HUB aborts if this requirement is ignored.

hub [-f -i -l -n -r -s -v -z] Input_file_name Accounting_file_name

f - Function names printed.
i - Interactive mode.
l - seLected list of function names printed.
n - graph Nodes' net values printed.
r - fault Reduction activated.
s - fault Simulator activated.
v - Verbose: print gBDD circuit data.
z - silos ii vector output.

Figure A.1: HUB's options and syntax.

Figure A.2 shows a portion of the accounting file's data output. Reasonable effort was made to provide uniqueness of information to allow filtering by means of the UNIX™ utilities *grep()* and *egrep()*. These were used extensively in preparing data for use by BBN's RS/1™ statistical analysis program.

HUB's data output.

Test vectors are generated in one of two formats: a time stamped (the vector has time related information included with it) version, Figure A.3a, is created with the *-z* option; otherwise the docuemented version of Figure A.3b is output which includes the graph node and pdcf identification numbers plus the input and output values. The documented vector is separated into 3 fields of information: (1) data on which graph node and which pdcf at that graph node for which the vector was created; (2) the primary input cube consisting of *1, 0s, and Xs*; (3) and the primary output cube which may have any value from the 5 value logic set *{0, 1, X, d, D}*.

```
Output_Test(11:1)    1011XXXXXX    D
Output_Test(12:0)    11010X0X0X    d
Output_Test(12:1)    11100X0X0X    d
Output_Test(12:2)    11110X0X0X    D
Output_Test(13:0)    110X010X0X    d
Output_Test(13:1)    110X100X0X    d
Output_Test(13:2)    110X110X0X    D
Output_Test(14:0)    110X0X010X    d
Output_Test(14:1)    110X0X100X    d
```

Figure A.3a: HUB's documented vector output.

```
100    1011XXXXXX
200    11010X0X0X
300    11100X0X0X
400    11110X0X0X
500    110X010X0X
600    110X100X0X
700    110X110X0X
800    110X0X010X
900    110X0X100X
```

Figure A.3b: HUB's SILOS II™ vector output.

To facilitate the file parsing phase, the follwing fixed circuit net list format is

used. The inputs and outputs may use either one of the formats, and may be a mixture for the same file. All statements start with a *space*. The order in which the netlist is written is important; failure to respect this sequence may result in unpredictable behavoir. The elements, except the inverter and buffer functions, must have a minimum of 2 inputs. (This required that the ISCAS list be modifed since they use 1-input NAND and AND gates to achieve these functions) Figure A.5 shows a

```
IN/input1, input2, ...//
OUT/output1, output2, ...//
ELEMENTs/input1, input2, .../output/
FOUT/input/output1, output2, output3/
END
```

Figure A.4: Neutral netlist syntax.

list of the elements, and the neutral netlist for a full adder is shown in Figure A.6. *Comments* may be used, and are indicated by a "*" in the first column of any line. There is also a *continuation* symbol, the "=" which must also be in the first column of any line.

```
AND
NAND
OR
NOR
NOT
BUF
```

Figure A.5: Permissible element list.

```
 IN/A1//                          ← Circuit's primary inputs.
 IN/B1//
 IN/CIN//
* INT/TMP//
 OUT/CARRY//                      ← Circuit's primary outputs.
 OUT/SUM//
*                                 ← A comment starts in col 1.
 OR/A11, B11/D/
 AND/A12, B12/E/                  ← Logic elements.
 AND/D, CIN1/H/
 NOR/H, E/J/
 NOT/J1/CARRY/
*
 OR/A13, B13, CIN2/F/
 AND/A14, B14, CIN3/G/
 AND/F, J2/K/
 NOR/K, G/L/
 NOT/L/SUM/
*
 FOUT/A1/A11, A12, A13, A14/
 FOUT/B1/B11, B12, B13, B14/
 FOUT/CIN/CIN1, CIN2, CIN3/
 FOUT/J/J1, J2/
 END                              ← Indicate end of netlist.
```

Figure A.6: Full adder netlist example.

10.0 Appendix B.

Figure B.1 describes the relation between a graph node's output names (its *lexemes)* and its graph node identification number as used by HUB. This list does not include the circuit's primary outputs since they are the first graph nodes in the graph node list maintained by HUB. All other graph nodes, that is the logic elements and the primary outputs, are listed. Figure B.2 shows some of the data HUB produces.

```
Program: node_lister
Date : Wed Oct 18 12:04:32 1989
Fichier d'entree : image.bdd
ready to go to hub_ATPG()

(Graph Node Number) : Net name

( 10) : GAT17
( 11) : GAT1
( 12) : GAT2
( 13) : GAT3
( 14) : GAT4
( 15) : GAT5
( 16) : GAT10
( 17) : GAT11
( 18) : GAT12
( 19) : GAT13
( 20) : GAT18
( 21) : GAT19
( 22) : GAT20
( 23) : GAT21
( 24) : GAT26
( 25) : GAT27
( 26) : GAT28
( 27) : GAT7
( 28) : GAT15
( 29) : GAT23
( 30) : GAT29
( 31) : GAT6
( 32) : GAT14
( 33) : GAT22
( 34) : GAT31
( 35) : GAT9
( 36) : GAT30
( 37) : GAT8
( 38) : GAT25
( 39) : GAT24
( 40) : TMR
```

Figure B.1: Listing of Image's graph node output names.

Date : Mon Nov 6 16:55:30 1989
Fichier d'entree : fadd.bdd

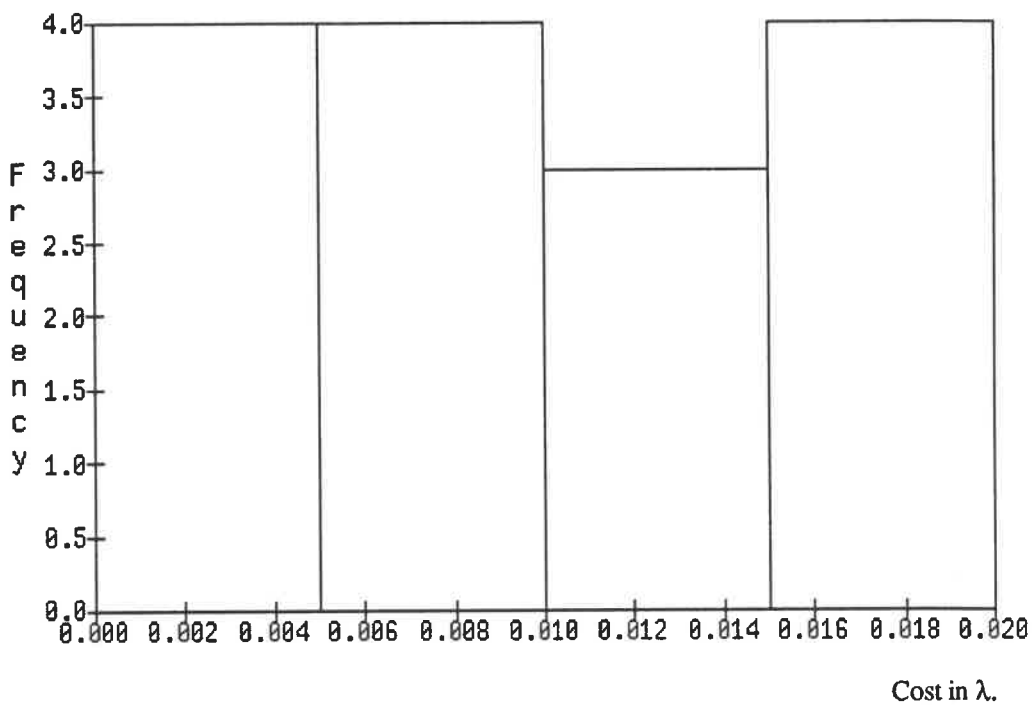
p/g	gn#	pdcf#	t_prop	t_just	t_bktracE	t_BacktraK	t_impl	t_time
P	3	0	0.00000000	0.00166660	0.00000000	0.00000000	0.00000000	0.00166660
P	3	1	0.00000000	0.00166660	0.00000000	0.00000000	0.00000000	0.00166660
G	3	2	0.00000000	0.00166660	0.00000000	0.00000000	0.00000000	0.00166660
P	4	0	0.00000000	0.00166660	0.00000000	0.00000000	0.00166660	0.00333320
P	4	1	0.00000000	0.00166660	0.00000000	0.00000000	0.00166660	0.00333320
G	4	2	0.00000000	0.00166660	0.00000000	0.00000000	0.00166660	0.00333320
P	5	0	0.00000000	0.00333320	0.00000000	0.00000000	0.00166660	0.00499980
G	5	2	0.00000000	0.00333320	0.00000000	0.00000000	0.00166660	0.00499980
P	6	1	0.00000000	0.00333320	0.00166660	0.00000000	0.00166660	0.00666640
G	6	2	0.00000000	0.00333320	0.00166660	0.00000000	0.00166660	0.00666640
G	7	3	0.00000000	0.00333320	0.00166660	0.00000000	0.00166660	0.00666640

budget	Flts	BK	Jbk	Pbk	Ibk	Bbk	Function	Pr	Sec	Cause
10.00000000	1	0	0	0	0	0	OR2	-	-	-999
10.00000000	1	0	0	0	0	0	OR2	-	-	-999
20.00000000	2	0	0	0	0	0	OR2	-	-	-999
10.00000000	1	0	0	0	0	0	AND2	-	-	-999
10.00000000	1	0	0	0	0	0	AND2	-	-	-999
20.00000000	2	0	0	0	0	0	AND2	-	-	-999
10.00000000	1	0	0	0	0	0	AND2	-	-	-999
20.00000000	2	0	0	0	0	0	AND2	-	-	-999
0.00000000	0	0	0	0	0	0	NOR2	-	-	-999
20.00000000	2	0	0	0	0	0	NOR2	-	-	-999
30.00000000	3	0	0	0	0	0	OR3	-	-	-999

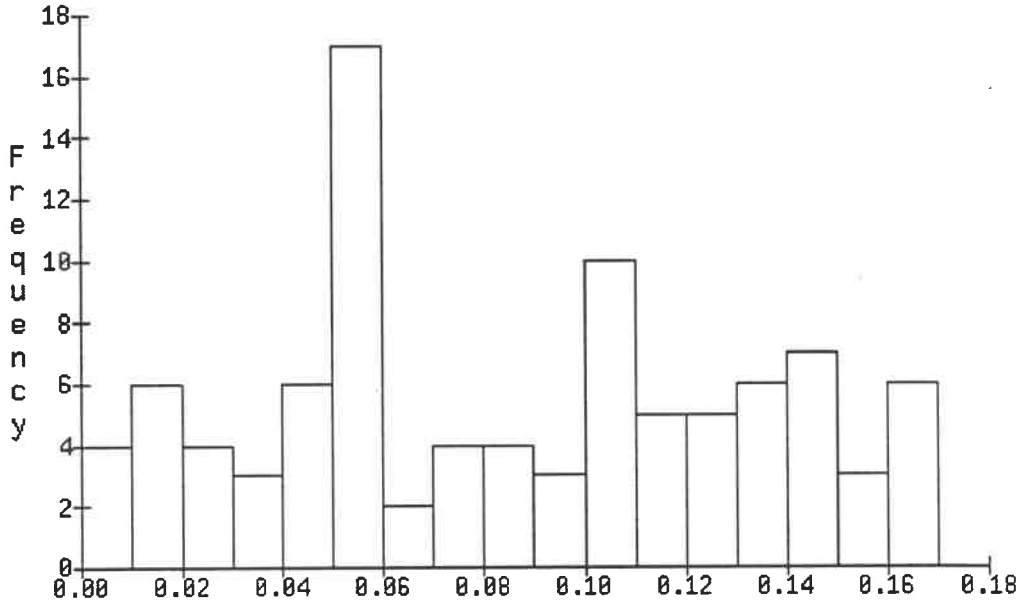
Figure B.2: Example of HUB's accounting data.

11.0 Appendix C.

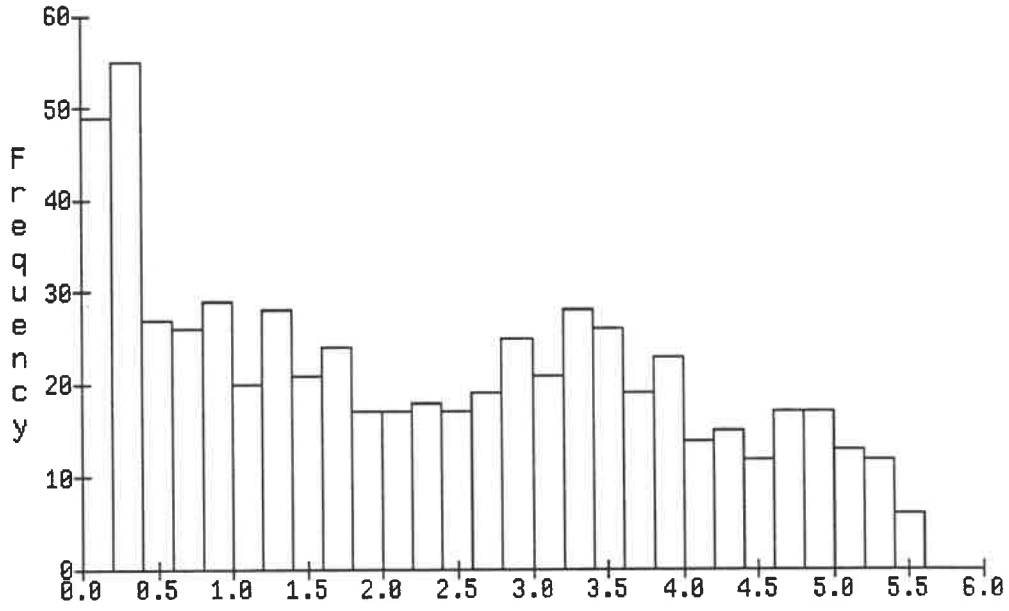
The following 6 histograms are the total cost histograms for the ISCAS circuits (C17, C95, C880, C1908, and C2670) and for the full adder cell. The cost is expressed in λ, the basic time units used by HUB. These histograms provide the visualization of the tabular data presented in the general results section of Chapter 6.



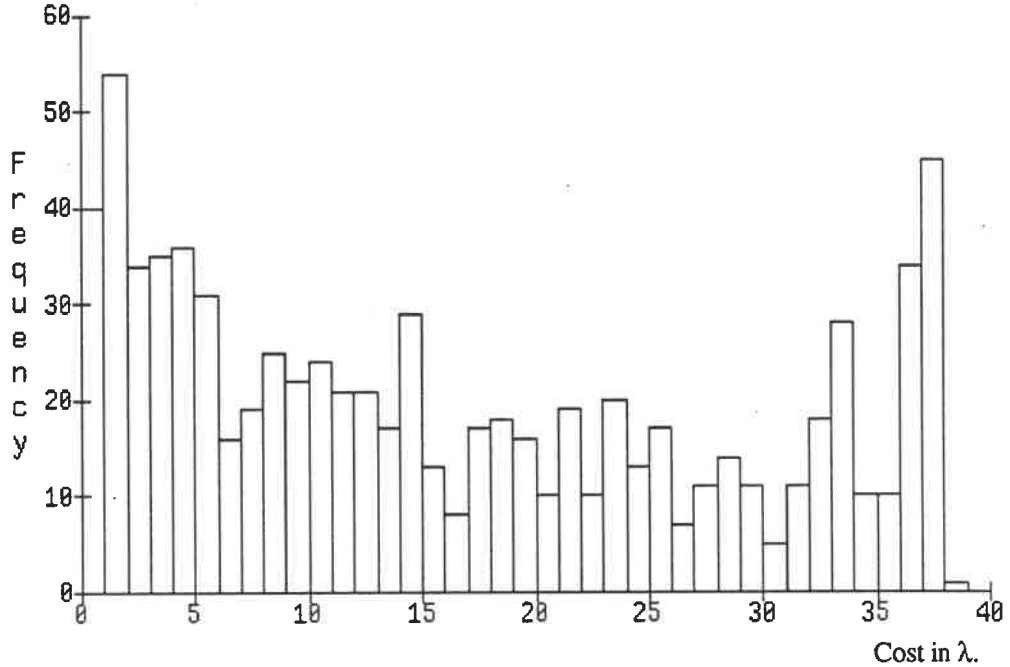
Graph C.1: C17 total cost histogram.



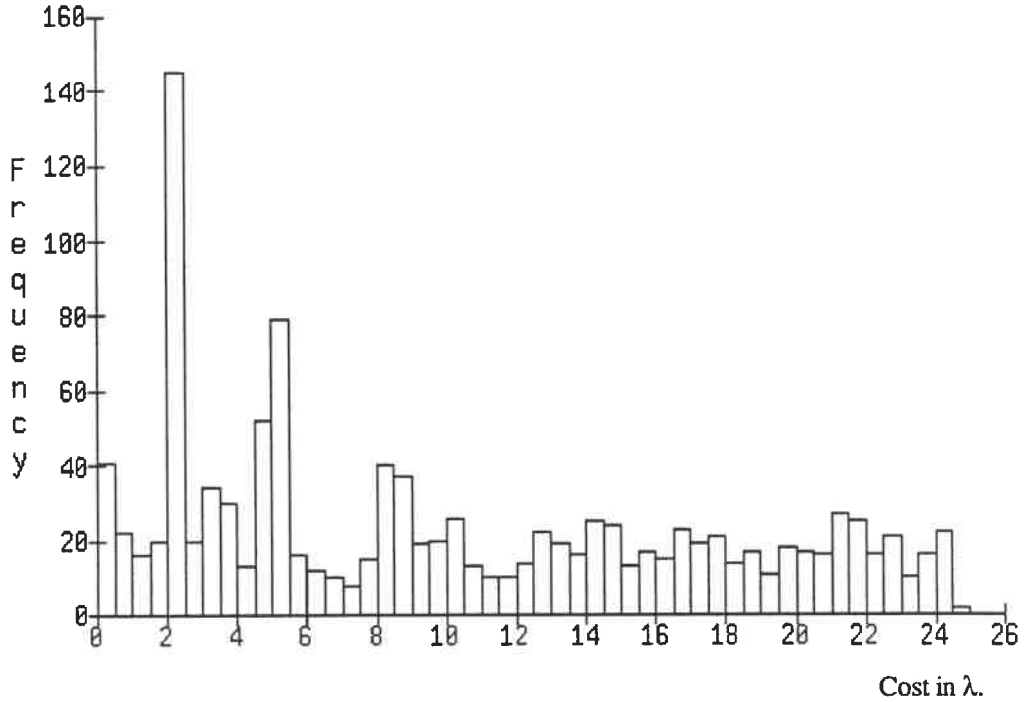
Graph C.2: C95 total cost histogram.



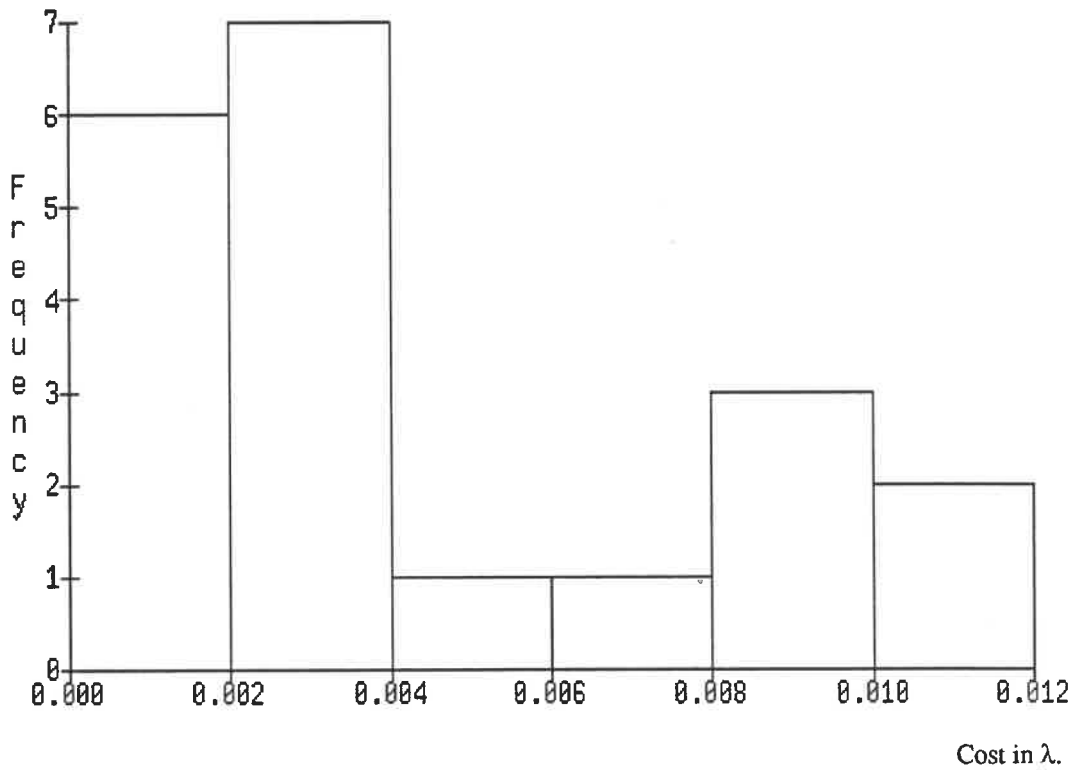
Graph C.3: C880 total cost histogram.



Graph C.4: C1908 total cost histogram.



Graph C.5: C2670 total cost histogram.



Graph C.6: Full adder total cost histogram.

ÉCOLE POLYTECHNIQUE DE MONTRÉAL
3 9334 00290864 6