1

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No. 2)

National Library
of Canada

Bibliothèque nationale
du Canada

Acquisitions and
Bibliographic Services Branch

Direction des acquisitions et
des services bibliographiques

395 Wellington Street
Ottawa, Ontario
K1A 0N4

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file - Votre référence

Our file - Notre référence

NOTICE

AVIS

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

If pages are missing, contact the university which granted the degree.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

Geometric Characterizations of Fault Patterns in Linear Systolic Arrays

by

Jiajun Ren

A thesis submitted to the
Faculty of Graduate Studies and Research
in the partial fulfillment of the requirements
for the degree of

Master of Computer Science

Ottawa-Carleton Institute for Computer Science
School of Computer Science
Carleton University
Ottawa, Ontario, Canada

January, 1994

©Jiajun Ren 1994

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

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 0-315-89811-9

Canada

Name JIAJUN REN

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

COMPUTER SCIENCE

SUBJECT TERM

0984 U·M·I

SUBJECT CODE

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS

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

EDUCATION

General 0515
Administration 0514
Adult and Continuing 0516
Agricultural 0517
Art 0273
Bilingual and Multicultural 0282
Business 0688
Community College 0275
Curriculum and Instruction 0727
Early Childhood 0518
Elementary 0524
Finance 0277
Guidance and Counseling 0519
Health 0680
Higher 0745
History of 0520
Home Economics 0278
Industrial 0521
Language and Literature 0279
Mathematics 0280
Music 0522
Philosophy of 0998
Physical 0523

Psychology 0525
Reading 0535
Religious 0527
Sciences 0714
Secondary 0533
Social Sciences 0534
Sociology of 0340
Special 0529
Teacher Training 0530
Technology 0710
Tests and Measurements 0288
Vocational 0747

LANGUAGE, LITERATURE AND LINGUISTICS

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

PHILOSOPHY, RELIGION AND THEOLOGY

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

SOCIAL SCIENCES

American Studies 0323
Anthropology
Archaeology 0324
Cultural 0326
Physical 0327
Business Administration
General 0310
Accounting 0272
Banking 0770
Management 0454
Marketing 0338
Canadian Studies 0385
Economics
General 0501
Agricultural 0503
Commerce-Business 0505
Finance 0508
History 0509
Labor 0510
Theory 0511
Folklore 0358
Geography 0366
Gerontology 0351
History
General 0578

Ancient 0579
Medieval 0581
Modern 0582
Black 0328
African 0331
Asia, Australia and Oceania 0332
Canadian 0334
European 0335
Latin American 0336
Middle Eastern 0333
United States 0337
History of Science 0585
Law 0398
Political Science
General 0615
International Law and Relations 0616
Public Administration 0617
Recreation 0814
Social Work 0452
Sociology
General 0626
Criminology and Penology 0627
Demography 0938
Ethnic and Racial Studies 0631
Individual and Family Studies 0628
Industrial and Labor Relations 0629
Public and Social Welfare 0630
Social Structure and Development 0700
Theory and Methods 0344
Transportation 0709
Urban and Regional Planning 0999
Women's Studies 0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

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

EARTH SCIENCES

Biogeochemistry 0425
Geochemistry 0996

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

HEALTH AND ENVIRONMENTAL SCIENCES

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

Speech Pathology 0460
Toxicology 0383
Home Economics 0386

PHYSICAL SCIENCES

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

Applied Sciences

Applied Mechanics 0346
Computer Science 0984

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

PSYCHOLOGY

General 0621
Behavioral 0384
Clinical 0672
Developmental 0620
Experimental 0623
Industrial 0624
Personality 0625
Physiological 0989
Psychobiology 0349
Psychometrics 0632
Social 0451

The undersigned recommend to the Faculty of Graduate Studies and Research
acceptance of the thesis

**Geometric Characterizations of Fault Patterns
in Linear Systolic Arrays**

submitted by Jianjun Ren, B.Sc.

in partial fulfilment of the requirements for the degree of Master of Computer Science

Thesis supervisor

Thesis supervisor

Director, School of Computer Science

Carleton University

January 1994

Abstract

Fault tolerance through the incorporation of redundancy and reconfiguration is quite common. In a redundant linear array of processing elements, k redundant links of fixed lengths are provided to each element of the array in addition to the regular links connecting neighbouring processors. The redundant links may be activated to bypass faulty processors. The number and distribution of faults can have severe impact on the effectiveness of such a method. Previous work in this area mainly focuses on fault patterns with exactly g_k number of faults, where g_k is the length of the longest bypass link. This thesis studies fault patterns with arbitrary number of faults by presenting geometric characterizations of fault patterns in linear systolic arrays. Based on the geometric characterizations, this thesis studies the problem of determining whether a given fault pattern with arbitrary number of faults is catastrophic. A necessary and sufficient condition for a fault pattern to be catastrophic is derived. The thesis proposes an efficient algorithm which can determine whether a fault pattern is catastrophic. The algorithm requires time $O(kn)$, where k is the number of bypass links and n is the number of faults.

Acknowledgment

I would like to express my sincere gratitude to my thesis supervisor, Prof. Nicola Santoro, for his constant help and guidance throughout the process of this research work. I am also very grateful to my thesis co-supervisor, Dr. Amiya Nayak, for the time he spent working with me and his valuable guidance.

I am grateful to my family members who showed their support in my efforts with continued interest and encouragement.

I thank all my thesis committee members, Professor Danny Krizanc, Professor J. Urrutia.

Table of Contents

Chapter 1 Introduction	1
1.1 Overview	1
1.2 The Problem, The Approach and The Results	5
1.2.1 The Problem	5
1.2.2 The Approach and The Results	5
1.3 Outline of the Thesis	9
Chapter 2 Background	10
2.1 Introduction	10
2.2 Systolic Arrays	11
2.3 Fault-Tolerant Systolic Arrays	13
2.3.1 Redundancy Schemes	14
2.3.2 Reconfiguration Schemes	16
2.4 Existing Characterization of Catastrophic Fault Patterns	20
2.4.1 Catastrophic Fault Patterns	20
2.4.2 Characterization of Catastrophic Fault Patterns	22
2.5 Testing in Systolic Arrays	25
2.5.1 Existing Testing Scheme 1	26
2.5.2 Existing Testing Scheme 2	29
2.5.3 Existing Testing Scheme 3	34
2.6 Chapter Summary	37
Chapter 3 Geometric Characterizations of Fault Patterns	38
3.1 Introduction	38
3.2 A Boolean Matrix Representation of Fault Patterns	39

3.3 Geometric Objects	42
3.3.1 Rectangle Type of Geometric Objects	42
3.3.2 Staircase Type of Geometric Objects	50
3.3.3 Segment Type of Geometric Objects	60
3.4 Chapter Summary	63
Chapter 4 Testing Scheme Using Rectangle and Staircase Objects	64
4.1 Introduction	64
4.2 A Testing Scheme	65
4.2.1 A Necessary and Sufficient Condition	65
4.2.2 A Testing Algorithm	67
4.3 Alternative Approach	75
4.4 Chapter Summary	76
Chapter 5 Testing Scheme Using Segment Objects	77
5.1 Introduction	77
5.2 A Necessary and Sufficient Condition	78
5.3 A Testing Algorithm	81
5.3.1 Gaps and Shadows	81
5.3.2 The Algorithm	84
5.4 Chapter Summary	95
Chapter 6 Summary and Conclusion	96
6.1 Summary of Main Contribution	96
6.2 Applications	98
6.3 Open Problems and Area of Future Works	99

List of Figures

1.1(a)	A fault pattern for links {1,4}	6
1.1(b)	A Boolean matrix	7
1.1(c)	The dead rectangles	7
2.1	A one-dimensional systolic array of processors	11
2.2	A two-dimensional systolic array of processors	12
2.3	Interstitial redundancy scheme	15
2.4	WRY/CB scheme	18
2.5	SCB scheme	19
2.6	A linear array of PEs	20
2.7	A fault pattern of size 5 with link redundancy {1,4}	21
2.8	A disconnected array of PEs	22
2.9	Boolean Matrix of A fault Pattern	29
2.10	A testing algorithm	36
3.1	A fault pattern of 5 faults with link redundancy {1,4}	39
3.2	A disconnected array of PEs	39
3.3	A Boolean matrix of a catastrophic fault pattern of 5 faults with link redundancy {1,4}	41
3.4	A dead rectangle	43
3.5	A rectangle is not dead	43
3.6	Two faulty chains	44
3.7	Two rectangles formed by two faulty chain	45
3.8	Dead elements and live elements	46
3.9	A dead row	47
3.10	$C=A \oplus B$	47

3.11	B intersects A at the right bottom vertex of A	48
3.12	Referring to Property 3.5	49
3.13(a)	A set of faults	49
3.13(b)	Identifying all faulty chains	50
3.13(c)	Building dead rectangles according Property 3.2	50
3.13(d)	Building dead rectangles according to Properties 3.4 and 3.5	50
3.14	An example of a staircase formed by two intersecting rectangles	51
3.15	An example of Algorithm 3.3	55
3.16(a)	The new staircase is exactly the union of two intersecting staircases	55
3.16(b)	New area is at the starting area of S and S'	56
3.16(c)	New area is in the middle area of S and S'	56
3.16(d)	The new area is in the ending area of S and S'	56
3.17(a)	Starting from a set of faults in a matrix	58
3.17(b)	Identifying all faulty chains according to the definition of faulty chain	58
3.17(c)	Identifying dead rectangles from faulty chains according to Property 3.2	59
3.17(d)	Expanding dead rectangles and identify all dead staircases according to Priorities 3.4, 3.5 and the definition of dead staircase	59
3.18	Examples of dead elements in a matrix where $G=\{1,3,5,7\}$	61
3.19(a)	A set of faulty elements in a matrix	61
3.19(b)	Identifying dead segments on the first row which are faulty elements	62
3.19(c)	Identifying dead segments on each row according to Definition 3.11	62
4.1(a)	Boolean matrix of a fault pattern	70

4.1(b)	Intermediate result of Algorithm 4.1	70
4.1(c)	Intermediate result of Algorithm 4.1	71
4.1(d)	Intermediate result of Algorithm 4.1	71
4.1(e)	The algorithm stops when a dead row is found	71
4.2(a)	Boolean matrix of a fault pattern	72
4.2(b)	Intermediate result of Algorithm 4.1	72
4.2(c)	Intermediate result of Algorithm 4.1	73
4.2(d)	Intermediate result of Algorithm 4.1	73
4.2(e)	No dead row is found	73
5.1	Claim 5.1 proof, Case 1, $j=1$	79
5.2	Claim 5.1 proof, Case 2, $j>1$	79
5.3	Theorem 5.2	82
5.4	Theorem 5.2 proof	82
5.5	An example of Theorem 5.2	83
5.6	The flow chart of Algorithm 5.1	86
5.7(a)	Boolean matrix of a fault pattern	90
5.7(b)	Two separated fault patterns generated	90
5.7(c)	The dead segments generated from the continuous faults in F_1	90
5.7(d)	F_1 is not catastrophic	91
5.7(e)	A dead segment of g_k elements is encountered	91

Chapter 1

Introduction

1.1 Overview

In today's VLSI technologies, it is economically feasible to use a massively parallel processing array, which consists of a large number of fine-grained cells (i.e., processing elements) for achieving high performance. A systolic array is an array of processors which rhythmically compute and pass data through the array. In a systolic array, large number of simple homogeneous processing elements (PEs) are connected in a regular fashion to give one or higher dimensional structures. Systolic arrays are characterized by their regular data flow, with streams of data items flowing through the structure synchronously in a fixed regular pattern. All communications take place between nearest neighbors.

In a systolic array fault are common - both PEs and links can be faulty. A common approach for achieving fault tolerance in VLSI-based systolic arrays is through the incorporation of *redundancy*, both in terms of redundant PEs and redundant links. The redundant PEs are used to replace any faulty PE(s); the redundant links are used to bypass the faulty PEs and reach the redundant PEs used as a replacement. The *process* of mapping faulty elements to spare (redundant) elements is called *reconfiguration*.

Intuitively, a system incorporating a large number of redundant PEs and 'long' redundant links should be able to tolerate a large number of PE failures. A long redundant link can bypass a large block of consecutive faulty PEs. However, long links are not always viable in such a system due to layout constraints. A long link will introduce larger propagation delays which could create synchronization problems and become the limiting factor in the performance of the system. Furthermore, note that to increase the number of redundant PEs in a chip requires an extra overhead of interconnections and switching circuitry which implies a higher likelihood of failure.

The effectiveness of using structural redundancy to increase fault-tolerance clearly does not depend solely on the amount of redundancy. In fact, the availability of a large number of redundant PEs is useless if these PEs cannot be successfully employed to replace the faulty ones. Thus, a main measure of fault-tolerance in such redundant arrays is the *reconfiguration capability* (or *reconfiguration effectiveness*); that is, the ability to map faulty elements to spare (using bypass links) while preserving the high degree of regularity and locality of reference required by the system to perform correctly.

It is therefore not surprising that a large amount of research has been devoted to the design of reconfiguration algorithms for redundant arrays as well as proposing redundant architectures which facilitate the reconfiguration process [7, 9, 11, 14, 19, 26, 28, 29].

An important (but often neglected) consideration is that the knowledge about the application field of the system can be usefully incorporated in the design process to obtain improved fault tolerance and performance. In particular, knowledge of the types and distribution of the faults, if properly employed, can make the design resilient to these

faults in the application domain; furthermore, the knowledge of the application field can significantly reduce the amount of testing needed to access the design. It is therefore important to acquire and integrate in the design the knowledge of the faults that occur, the fault distribution, etc. [33]. These data should be used (together with the circuit details) to assess fault tolerance.

The effectiveness of using redundancy to increase fault tolerance in a regular architecture clearly depends on both the amount of redundancy and the reconfiguration capability of the system. It does however depend also on the distribution of the faults in the system. A fault pattern is a distribution of a certain number of faults. In fact, faults patterns in a systolic array may have catastrophic effect on the entire structure and cannot be overcome by any amount of clever design. A fault pattern is said to be catastrophic if the removal of the faulty PEs together with their incident links disconnects the input from output. More precisely, it will create a situation in which it will not be possible to reconfigure the array and carry on with normal operation.

Fault detection capability is usually built into the system in which each PE performs built-in self-test and makes its result visible. Through this process, it is known which PEs and links are faulty and which are not. To our knowledge, no other capability that can assess the net effect of a set of faulty components is provided with the system once it is manufactured. To ensure continuous operation, it is necessary to have a mechanism that can test if the occurrence of a pattern of faults will have catastrophic effect on the system.

A large amount of research [1, 2, 30, 34, 35, 36, 38, 39, 40, 41] has been done to characterize catastrophic fault patterns. The characterization has revealed several interesting properties about catastrophic fault patterns. It has been shown that the

minimum number of faults needed to construct a catastrophic fault pattern is not a function of the total number of bypass links but of the length of the longest bypass link. For this reason, the researchers have considered faulty patterns which have exactly the same number of faults as the length of the largest bypass link. Based on the characterization of catastrophic fault patterns, testing schemes have been derived to test whether or not a given fault pattern is catastrophic.

In particular, the case of 2-link redundancy has been studied in [1, 2, 34, 35, 38, 39, 40, 41]. The researchers studied the recognition, enumeration and construction of catastrophic fault patterns. The case of arbitrary link redundancy has been studied in [36], where the problem of determining whether or not a given fault pattern is catastrophic has been studied.

1.2 The Problem, The Approach and The Results

1.2.1 The Problem

Most of the previous work have been focused on the fault patterns with exactly g_k faults, where g_k is the length of the longest bypass link. However, the number of faults practically can be arbitrary. Therefore, the study of the more general case where the number of faults is arbitrary is required. This study will involve two aspects. One is a characterization that can deal with the case of arbitrary number of faults, especially the case where the number of faults is greater than g_k ; the other is a testing mechanism which can test arbitrary fault patterns for systems with arbitrary redundant links. The main objective of this thesis is to study the characteristics of fault patterns whose number of faults is arbitrary. Then, based on the characterization, the thesis will derive a testing mechanism to decide whether a fault pattern is catastrophic.

1.2.2 The Approach and The Results

Fault patterns with arbitrary number of faults are studied by mapping a given fault pattern into a Boolean matrix, in which the faults are treated as geometric objects which are functionally "dead". By "dead" we mean that the PE's corresponding object is either faulty or non-faulty but useless for all functional purposes. Objects are combined in a systematic way to obtain larger objects.

After the given fault pattern is mapped into a Boolean matrix, we study the faults, particularly their impact on their neighbouring processing elements. If some of the neighbouring processing elements become useless for all practical purposes, we combine the original faults and those neighboring elements to form geometric dead objects. We then study the impact on the neighbouring elements of these objects.

Through this process, we continue to form larger and larger geometric dead objects, such as dead rectangles and dead staircases, in the Boolean matrix. The result of this process is a Boolean matrix containing a set of geometric dead objects which cannot be further enlarged. The rest of the Boolean matrix corresponds to all the non-faulty and useful elements which can be employed by reconfiguration schemes.

Example 1.1 Figure 1.1(a) shows a linear unidirectional array in which each PE is connected to its immediate neighbors and to the PEs at distance 4. The faulty PEs (faults) are shadowed. The fault pattern is mapped into a Boolean matrix, as shown in Figure 1.1(b), in which "1" stands for the faulty element and "0" stands for the non-faulty element. Elements (1,1), (1,3) and (2,2) are faulty, and thus are dead. Elements (1,2), (2,3) are useless in any computation. We now form a larger dead rectangle which is $\{(1,2), (1,3), (2,2), (2,3)\}$. The dead rectangles are shown in Figure 1.1(c). All the elements which are not in dead rectangles can be employed by any reconfiguration scheme.

Figure 1.1(a): A fault pattern for links {1,4}

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

Figure 1.1(b): A Boolean matrix

$$\begin{bmatrix} \boxed{1} & \boxed{0} & 0 \\ 0 & \boxed{1} & 0 \end{bmatrix}$$

Figure 1.1(c): The dead rectangles

Following this approach, we characterize faulty patterns of arbitrary size for linear arrays first for the case of 2-link redundancy and then for the arbitrary link redundancy.

Based on the geometric characterization briefly described above, we can easily solve the problem of deciding whether a given fault pattern is catastrophic. We prove that a given fault pattern is catastrophic if any geometric dead object covers every column of the Boolean matrix generated from the fault pattern.

Following the same approach, we study the more general case of arbitrary link redundancy; that is, each PE in the array connected to k different PEs. As usual, we start from faults, build geometric dead objects, in this case, dead segments. We could build all dead segments in the Boolean matrix corresponding to a given fault pattern. The rest of the Boolean matrix are all non-faulty and useful elements which will be employed by reconfiguration schemes. Based on this geometric characterization, we can also easily solve the problem of deciding whether a given fault pattern is catastrophic. We prove that a given fault pattern is catastrophic if at least a dead segment covers every column of the Boolean matrix. We develop a testing algorithm which can decide whether or not a

given fault pattern is catastrophic by building dead segments and in the mean time checking if a dead segment covers every column of the Boolean matrix. The algorithm requires $O(kn)$ time where k is the number of links and n is the number of faults PEs. This improves on the existing $O(kn \log k)$ bound recently established [48].

The main contributions of this thesis are: 1) geometric characterizations for arbitrary fault patterns in a linear array with arbitrary link redundancy; 2) testing schemes, each dealing with a different geometric object, for testing if a given fault pattern is catastrophic; 3) an improved upper bound on the complexity of testing.

1.3 Outline of the Thesis

The thesis is organized as follows. In Chapter 2, a background on systolic arrays and fault tolerance, as well as fault-tolerant systolic arrays is given; also given in this chapter is a general description of a known characterization of catastrophic fault patterns in the presence of g_k faults.

Chapter 3 contains one geometric characterization of fault patterns where the links are $\{1, g_k\}$. Then the more general case where the links are $\{g_1, g_2, \dots, g_k\}$ is dealt with. The other geometric characterization of catastrophic fault patterns is presented.

Based on the first geometric characterization, a necessary and sufficient condition for a fault pattern to be catastrophic for 2-link redundancy is given in Chapter 4. A testing algorithm to decide if a fault pattern is catastrophic is also developed in Chapter 4.

Based on the second geometric characterization, a necessary and sufficient condition for a fault pattern to be catastrophic for k -link redundancy is given in Chapter 5. A testing algorithm to decide if a fault pattern is catastrophic is also developed in Chapter 5.

The major results are summarized in Chapter 6. Some open problems are listed, and a few conclusions that might serve as guidelines for future research are given.

Chapter 2

Background

2.1 Introduction

Fault tolerance is an important factor in systolic arrays and has received considerable attention in recent years. This chapter provides an overview of systolic arrays, fault-tolerant systolic arrays in Section 2.2 and Section 2.3 respectively.

Fault patterns, whose occurrence has catastrophic effects on the system, were studied thoroughly [1, 2, 34, 38]. A complete characterization of catastrophic fault patterns for a certain number of faults was developed [34]. These previous results are given in Section 2.4. Given a fault pattern, it is crucial, from reconfiguration point of view, to know whether the fault pattern is catastrophic. Section 2.5 will give three existing testing schemes which can check the catastrophe of a given fault pattern using different approaches.

2.2 Systolic Arrays

A systolic system is a network of processors which rhythmically compute and pass data through the system. Systolic arrays are readily amenable to VLSI implementation. It is particularly suitable to a special class of computation bound algorithms and takes advantage of their regular, localized data flow. The basic principle of systolic design is that all the data, while being pumped regularly and rhythmically across the array, can be effectively used in all the processing elements. The systolic array features the important properties of modularity, regularity, local interconnection, a high degree of pipelining, and highly synchronized multiprocessing.

Systolic arrays are effective means to achieve speedups in execution time for computation-bound problems by exploiting parallel processing and pipelining. A large number of simple homogeneous processing elements (PEs) can be connected in a regular fashion to give one or higher dimensional structures. Systolic arrays are characterized by their regular data flow, with streams of data items flowing through the structure synchronously in a fixed regular pattern. All communications take place between nearest neighbors. An example of a one-dimensional systolic array is given in Figure 2.1. Another example of two-dimensional systolic array is presented in Figure 2.2.

Figure 2.1: A one-dimensional systolic array of processors

Figure 2.2: A two-dimensional systolic array of processors

Array processors have been proposed in real-time control systems, signal and image processing systems to solve computation-intensive problems such as fast Fourier transformation, convolution, and matrix operations, which consist of high volume and iterative (or recursive) computation steps [21]. In today's VLSI technologies, it is economically feasible to use a massively parallel processing array, which consists of a large number of fine-grained cells (i.e., processing elements) for achieving high performance in such applications. Systolic arrays and massively parallel processor (MPP) are example of such VLSI arrays. For a very large array, faults are very likely to occur during normal operations. Therefore, on-line fault-tolerance processes, which include fault detection, diagnosis, reconfiguration, and rollback error recovery are crucial for such systems.

2.3 Fault-Tolerant Systolic Arrays

The architecture described in the preceding section has the major drawback that a single processor or edge failure may render the entire network unusable if the algorithm running on the network requires that the topology of the network does not change. The failure of a single processor or the failure of a link between two processors would destroy the topology of this architecture. Thus, some form of fault tolerance must be incorporated into these architecture's in order to make the network of processors more reliable. The fact that this structure is highly regular makes it easy to achieve a high degree of fault tolerance through the use of redundant processors and links.

Using VLSI technology, systolic arrays are now being manufactured cost effectively due to their highly dense and efficient layout [15, 27, 31]. On one hand, it is now possible to incorporate a large number of processing elements and their necessary interconnections into a single VLSI/WSI chip. On the other hand, low yield of VLSI/WSI has been considered of a major concern. The low yield comes from both complex manufacturing processes and very high circuit density in VLSI/WSI components. Therefore, it is necessary to build redundancy and reconfiguration mechanisms into VLSI/WSI device for improving the yield.

Two different approaches to enhance low yield has been suggested [18, 33]. One uses proper design rules and advanced manufacturing techniques to reduce failures during manufacturing. The other approach use modular redundancy in the chip/wafer design so that the desired target system can be implemented by interconnecting the good modules

on the chip/wafer. The use of redundant modules in the chip/wafer to enhance yield is popular because the first approach is almost impossible to achieve in VLSI/WSI components with current manufacturing technology. Thus, redundancy and reconfiguration are two fundamental aspects considered in achieving fault tolerance. A lot of research has been done in the design [11, 17, 18, 20] and implementation [10, 15] of fault tolerant systolic architecture's.

2.3.1 Redundancy Schemes

Structure redundancy is implemented by adding redundant PEs and redundant interconnections to replace faulty PEs. A simple way to provide redundancy in systolic arrays is by adding spare rows and/or spare columns of PEs. The spare PEs need not be arranged that way; there are, however, other ways to arrange spare PEs in the array. An arrangement of spare PEs can be judged by many factors which include spare utilization, effectiveness of tolerating various fault distributions, the length of connecting wires after reconfiguration, etc. Some of the popular schemes will be described here.

In a "local (or interstitial) redundancy scheme,"[67], spare PEs are distributed uniformly into an array, and a small group of PEs (called a block) share the spare PEs which are in the block. In this approach, a faulty PE can be replaced only by a "local" spare PE, i.e., a spare in the same block. Since spare PEs are partitioned and distributed into each block of the array, maintaining the required interconnection structure becomes much simpler. It requires a simple reconfiguration control algorithm and circuit, and low hardware and time overhead. The length of connecting wires after reconfiguration is

limited by the width of a block. The drawback of this scheme is that the array may fail due to the lack of spare PEs in one block when other blocks contain spare PEs.

Example 2.1 An example of a systolic array using interstitial redundancy is shown in Figure 2.3.

Figure 2.3: Interstitial redundancy scheme

In a "global redundancy scheme", there is no restriction on the usage of spares to replace faulty PEs. In principle, any faulty PE can be replaced by any spare PE. The extent to which this is true depends on the flexibility and therefore the complexity of the adopted redundancy scheme.

In a "two-level redundancy scheme"[65], the array is partitioned into identical blocks with each block containing a fixed number of spare PEs. This provides the first level of redundancy, called local redundancy. The local redundancy is effective for the distributed

faults but it is not sufficient to overcome clustered faults. A second level of redundancy, called the global redundancy, is therefore provided by adding spare blocks to the array for tolerating failures of blocks induced by clustered faults. This redundancy scheme is called two-level redundancy. The two-level redundancy can be viewed as a compromise between the pure local and pure global redundancy scheme.

Some of the advantages of the two-level redundancy scheme over the local and the global redundancy scheme are the following[65]:

1. The two-level redundancy scheme has better spare utilization than that of global whole row (and/or column) bypass redundancy schemes.
2. The local redundancy is effective for distributed faults; the global redundancy is effective for clustered faults. A pure local redundancy cannot tolerate clustered faults.

2.3.2 Reconfiguration Schemes

Reconfiguration is defined as the operation of replacing faulty components with spares while maintaining the original interconnection structure.

Several reconfiguration strategies have been suggested to restore fault-free operation in regular processor arrays with faulty elements [7, 22, 54, 66]. Two of these approaches are briefly reviewed here. One is due to Rosenberg [54] and the other is due to Kung and Lam [22]. Rosenberg's approach, called the Diagnose approach, provides alternate paths between PEs. When a PE is diagnosed to be faulty, it is bypassed by using the alternate paths. The main feature of the Diagnose approach is its simplicity. However, a potential

disadvantage is that, in the presence of consecutive faulty PEs, logically connected PEs may be far apart, requiring the reduction of clock speed and thus reducing throughput of the array.

An alternate approach called systolic fault tolerance is due to Kung and Lam [22]. This approach uses delay registers in the alternate paths. When a PE is diagnosed to be faulty, it is bypassed through the bypass registers. Use of memories or registers in bypass paths helps to avoid increase in inter-PE wire length due to reconfiguration. The two reconfiguration methods described above make the assumption that the circuits in the alternate paths remain fault-free.

Since it is possible to map an arbitrary large algorithm to fixed-size array architecture, fault-tolerant schemes that provide graceful degradation should be considered. The idea is to dynamically reconfigure so that, after removal of faulty PEs, a fully operational, reduced size array is available. There are two types of reconfiguration schemes for using spare PEs in mesh arrays with global redundancy:

a) Whole row (and/or column) bypass (WRY/CB) [16]: In this type, a faulty PE causes its whole row or column to be bypassed. There are many different reconfiguration schemes that are based on this type. They have the advantage of requiring simple circuitry to bypass rows and/or columns. Their main drawback is poor utilization of spares. The problem of choosing the minimum number of spare rows and/or column that cover all the faulty PEs is NP-complete [25]. Thus, various heuristic reconfiguration algorithms are used.

Example 2.2 Figure 2.4 shows a WRY/CB reconfiguration of any array with one spare row and column and two faulty elements. The faulty element in column 1 causes column 1 to be bypassed, and the faulty element in row 2 causes row 2 to be bypassed.

Figure 2.4: WRY/CB scheme

b) Single-cell bypass (SCB) [42, 60]: In this reconfiguration scheme, only faulty PEs are bypassed. This scheme has better utilization of spare PEs. After the array is reconfigured, each PE of the array is mapped to a fault-free PEs in the wafer. In its most flexible form, mapping in the required structure depends only on the number of fault-free PEs and not on the location of the PEs in the wafer. Depending on the flexibility of the mapping, the reconfiguration algorithm can become very complex. In that case, many interconnecting lines and complicated switches are needed. In addition, there is a possibility of long interconnection wires after reconfiguration.

Example 2.3 Figure 2.5 shows the reconfiguration of an array with one spare row and one spare column with three faulty elements bypassed.

Figure 2.5: SCB scheme

2.4 Existing Characterization of Catastrophic Fault Patterns

2.4.1 Catastrophic Fault Patterns

The system being studied is one-dimensional (or linear) arrays as shown in Figure 2.6. The basic components of such a system are the *processing elements* (PEs) connected by unidirectional links; for convenience, it is assumed that the array is infinitive. There are two kinds of links in redundant arrays: *regular* or *bypass*. Regular links exist between neighboring PEs; bypass links connect non-neighboring PEs. The bypass links are used strictly for reconfiguration purposes when a fault is detected; in the absence of faults, the bypass links are redundant. Bypass links are not shown in Figure 2.4 for simplicity.

Figure 2.6: A linear array of PEs

A *fault* in a PE can be best described as its inability to perform correct operation on correct inputs, possibly due to some elementary circuit or gate-level failures which could be either permanent or transient. In this thesis, the focus of the investigation is on permanent faults. Any non-faulty PE which is not active during a computation is said to be a *spare* for that computation; thus, there is no a priori designation distinguishing

spares from other PEs. The location of spares is irrelevant for this discussion although it may play a significant role in real life situation.

Let $A = \{x_1, x_2, \dots, x_N, \dots\}$ denote one-dimensional array of PEs, where each $x \in A$ represents a processing element and there exists a direct link between x_i and x_{i+1} , $1 \leq i < \infty$. Any link connecting x_i and x_j where $j > i+1$ is said to be a bypass link. The length of a bypass link, connecting x_i and x_j , is the distance in the array between x_i and x_j ; i.e., $|j-i|$. Given an integer $g \in (1, n]$ and an array A of size N , A is said to have link redundancy, if for every $x_i \in A$ with $i \leq N-g$ there exists a link between x_i and x_{i+g} . The array has link redundancy $G = \{g_1, g_2, \dots, g_k\}$ where $g_j < g_{j+1}$ and $g_j \in (1, N]$, if A has link redundancy $g_1, g_2, \dots, g_k$.

In the following, it will be assumed that $g_1 = 1$ and no other link except the ones specified exists in the array; that is, G totally defines the link structure of A .

Given a linear array A , a *fault pattern* $F = \{f_1, f_2, \dots, f_n\}$ for A is a subset of A , where for every $f \in F$, x_f is faulty.

Example 2.4 Example of a fault pattern of 5 faults with link redundancy $\{1, 4\}$ is given in Figure 2.7.

Figure 2.7: A fault pattern of size 5 with link redundancy $\{1, 4\}$

A fault pattern is *catastrophic* for a linear array A of PEs with link redundancy G if the array cannot be reconfigured in the presence of the faulty PEs. That is, the removal of the faulty elements and their incident links will cause the array to be disconnected.

Example 2.5 The fault pattern given in Figure 2.8 is catastrophic since the removal of the faulty PEs and their incident links disconnect the array, as shown in Figure 2.8.

Figure 2.8: A disconnected array of PEs

2.4.2 Characterization of Catastrophic Fault Patterns

The following summarizes the existing characterization of catastrophic fault patterns[26].

F is catastrophic with respect to G implies that the *cardinality* of F , $|F| \geq g_k$. It illustrates that the cardinality of a catastrophic fault pattern does not depend on the number of bypass links available but only on the length of the longest bypass link.

The *width* W_F of a fault pattern (or fault window) is the number of PEs between and including the last fault in F . That is, if $F = \{f_1, f_2, \dots, f_m\}$ then $W_F = f_m - f_1 + 1$.

In the following, the width of the largest possible fault window is given for both unidirectional and bidirectional links.

Unidirectional Links: Let $|F|=g_k$ and $G=(g_1, g_k)$ be a set of unidirectional links. Then the width of the fault window for G is given by $W_F=(g_k-1)^2+1$.

Bidirectional Links: Let $|F|=g_k$ and $G=\{g_1, g_k\}$ be a set of bidirectional links. Then the width of the fault window for G is

$$W_F = \left(\left\lfloor \frac{g_k}{2} \right\rfloor + 1 \right) g_k + \left\lfloor \frac{g_k}{2} \right\rfloor + 1$$

Obviously, the width of the fault window varies with the size of G . Consider k , the size of G . When $k=1$, there are no bypass links; hence, any single fault is catastrophic. When $k=g_k$, only a cluster of g_k faults can be catastrophic. When $k < g_k$, let $G' \subseteq G$ and $W_{F'}$, W_F be the corresponding widest fault windows. Then $W_{F'} \geq W_F$.

Bounds on the number of catastrophic fault patterns for a given link configuration were established.

Let $G=\{1, g_k\}$. Then the number of catastrophic fault patterns Γ_G is given by

$$\Gamma_G = \sum_{i=1}^{\left\lfloor \frac{g_k}{2} \right\rfloor} \frac{1}{i+1} \binom{2i}{i} \binom{g_k-2}{g_k-2i}$$

Given an arbitrary G' , the number of catastrophic fault patterns $\Gamma_{G'}$ is given by

$$\Gamma_{G'} \leq \sum_{i=1}^{\left\lfloor \frac{g_k}{2} \right\rfloor} \frac{1}{i+1} \binom{2i}{i} \binom{g_k-2}{g_k-2i}$$

All these previous results dealt with the minimum fault patterns where the number of faults is exactly g_k , where g_k is the length of the longest bypass link. For any fault pattern, having at least g_k faults, is a necessary condition to be catastrophic. The previous work on the specific case where $|F|=g_k$ is not easily extendible to the general case where $|F| \geq g_k$. From a practical view point, the number of faults happening in a systolic array can be arbitrary. For any fault pattern, having at least g_k faults, is a necessary condition to be catastrophic. Therefore, we can ignore the case where $|F| < g_k$. For the case of $|F|=g_k$, as mentioned above, a complete characterization has been given. It is, however, not practically useful. It is obvious that a general characterization for the general case where number of faults is arbitrary is needed, especially testing of catastrophic fault patterns.

2.5 Testing in Systolic Arrays

The need for efficiency ways of testing complex system is widely recognized, and is as necessary for systolic arrays as it is for any other complex system. It is likely for a systolic array to have some defective PEs (faults) which can be tolerated. This implies that if these faults happen in the systolic array, data can always flow from input of the systolic array to the output bypassing all faults by employing some reconfiguration scheme. However, in some cases, the occurrence of some faults will result in that data cannot flow from the input of the systolic array to the output; in other words, the flow of data will always be stopped at some fault no matter what reconfiguration scheme is used. The fault patterns consisting of these faults, which have been introduced in Section 2.4, are catastrophic since any of these fault patterns has a catastrophic consequence to the systolic array.

It is crucial for a systolic array, from the reconfiguration point of view, to know whether a given fault pattern is catastrophic when the fault pattern happens. If a given fault pattern is not catastrophic, we can consider applying some reconfiguration scheme to bypass all the faults. There is no sense to apply some reconfiguration scheme to the systolic array in which a catastrophic fault pattern occurs. Therefore, the testing of catastrophe of a fault pattern serves as a basis for the later reconfiguration

In this section, three existing testing schemes using different approaches will be given.

2.5.1 Existing Testing Scheme 1

A testing algorithm has been described in [34] to determine if a given fault pattern is catastrophic for a 2-link redundant unidirectional linear array. In this algorithm, only the minimal fault patterns are considered, i.e., the fault patterns contain exactly g_k faults where g_k is the bypass link in the link configuration $G=\{1, g_k\}$.

First a given fault pattern is mapped into a Boolean matrix using the same representation technique described in Chapter 3. Then, from the Boolean matrix a string is derived on alphabet $\{(f), *\}$ corresponding to the given fault pattern. This alphabet consists of three symbols: an open parenthesis with weight f , ' $(f$ ', a close parenthesis with weight $)$ ' and a symbol '*'. A " $(f$ " denotes a block of f open parentheses. The derivation of such a string on $\{(f), *\}$ from the Boolean matrix W is outlined below.

A sequence $S_1, S_2, \dots, S_{|F|}$ of non-zero elements of W is constructed as follows:

- $S_1 = W[1, 1]$ and $S_{|F|} = W[1, g_k]$.
- If $S_l = W[i, j]$ then S_{l+1} is the sole non-zero element among $W[i-1, j+1]$, $W[i, j+1]$ and $W[k, j+1]$, $k > i$.

With this sequence, a string χ on the alphabet $\{(f), *\}$ is associated as follows. Given $S_l = W[i, j]$, $1 \leq l \leq |F|$. The string is defined as follows:

$$\chi(S_{l+1}) = \begin{cases} "*" & \text{if } S_{l+1} = W[i, j+1] \\ ")" & \text{if } S_{l+1} = W[i-1, j+1] \\ "(f" & \text{if } S_{l+1} = W[i+f, j+1], \quad f \geq 1 \end{cases}$$

The cost $c(S_i)$ of an element in the sequence is defined as follows

$$c(S_i) = \begin{cases} f & \text{if } \chi(S_i) = "(" \\ 0 & \text{if } \chi(S_i) = "*" \\ -1 & \text{if } \chi(S_i) = ")" \end{cases}$$

The resulting string χ associated to a catastrophic fault pattern F has the property of being *well-formed* in the following sense:

1. for any subsequence $S_1, S_2, \dots, S_j$, $1 \leq j \leq |F|$, starting with S_1 ,

$$\sum_{i=1}^j c(\chi(S_i)) \geq 0 \quad \text{i.e., } |L| \geq |R|, \quad \text{and}$$

where L is the number of open parentheses, R is the number of close parentheses,

2. for the entire sequence $S_1, S_2, \dots, S_{|F|}$,

$$\sum_{i=1}^{|F|} c(\chi(S_i)) = 0.$$

As before, the "*" symbol can occur anywhere in χ except in $\chi(S_1)$ and $\chi(S_{|F|})$.

An example of a well-formed string on $\{(f, *) : f \in \mathbb{Z}\}$ is $"(4)))(2)*))"$. In this case, the sum of the weights on all open parentheses is at least the number of close parentheses at any point in the string, and the sum of weights on all open parentheses is equal to the

number of close parentheses for the entire string. An example of a string on $\{(f_i) : f_i \in \mathbb{Z}\}$ which is not well-formed is $"(2)))(3)*)")"$.

Any such well-formed string corresponds to a catastrophic fault pattern. This fact is stated in the following theorem which constitutes a necessary and sufficient condition for a fault pattern to be catastrophic.

Theorem 2.1 [34] A fault pattern is catastrophic if and only if χ is well-formed, where χ is the string associated with the fault pattern.

Algorithm 2.1 [34] The testing scheme is based on Theorem 2.1. As the first step of the algorithm, the given fault pattern is first mapped onto the Boolean matrix W . In the next step, each entry of W is verified to see if it satisfies the well-formed property. If there is no violation, then the given fault pattern F is catastrophic; otherwise, F is not catastrophic.

The algorithm is proved to have $O(|F|)$ time and space.

Example 2.6 The following example deals with the existing testing scheme to determine if a fault pattern is catastrophic.

Consider a Boolean matrix representation of a fault pattern given in Figure 4.1. To test if it is catastrophic for the link redundancy $G=\{1,10\}$ in which the links are unidirectional, the corresponding string of weighted parentheses is $"((*)(3)"$ which is not well-formed. Therefore, the given fault pattern is not catastrophic.

As long as the number of faults is g_k or less, the testing algorithm is very efficient. In practice, the number of faults can be larger than g_k , in which case the existing algorithm cannot be used. It is not known if the existing algorithm can be modified to deal with any arbitrary fault pattern whose number of faults is greater than g_k . Therefore, a testing scheme that can handle fault patterns with arbitrary number of faults is needed. A testing scheme to deal with fault patterns with arbitrary faults is presented in the following section.

f_1								f_2		f_3
1	0	0	0	0	0	0	0	1	0	1
0	1	0	1	0	0	1	0	0	0	0
0	0	1	0	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0
										f_{10}

Figure 2.9: Boolean matrix of a fault pattern

2.5.2 Existing Testing Scheme 2

In this section, the testing scheme of [36], for a fault pattern to be catastrophic in a linear unidirectional array with k -link redundancy is described. First some fundamental properties which any catastrophic fault pattern must satisfy are given. These properties together constitute a necessary and sufficient condition for a fault pattern to be

catastrophic for k -link redundant system. The testing algorithm whose time complexity is $O(kg_k)$ is based on this necessary and sufficient condition.

Again the given fault pattern is mapped onto a Boolean matrix W . The researchers consider only the minimal fault pattern, which is a fault pattern whose number of faults is exactly g_k . Once the mapping is done, the interior, exterior and border of the fault pattern mapped in W are identified as follow:

Notice that the row coordinates of F is an ordered set $\{x_0, x_1, \dots, x_{g_k-1}\}$ of row indices of W corresponding to the faults f_i ($0 \leq i \leq g_k-1$). Consider a fault f_i whose location is $W[x_i, y_i]$. Now, the location $W[i, y_i]$, with respect to f_i , is interior if $i < x_i$, border if $i = x_i$, and exterior if $i > x_i$.

For a given fault pattern F , $I(F)$ (i.e., interior of F) is the set of all interior elements, $B(F)$ (i.e., border of F) is the set of all border elements, and $E(F)$ (i.e., exterior of F) is the set of all exterior elements.

Now with respect to the matrix representation F , a fault pattern F is catastrophic for an array A with link redundancy G if it is not possible to reach any exterior element from any interior element using links in G . The area A_F of a fault pattern F is the number of interior and border elements; that is,

$$A_F = \|I(F) \cup B(F)\| = \sum_{j=0}^{g_k-1} (x_j - 1).$$

A special catastrophic fault pattern, called "reference fault pattern" (RFP), is known to be unique for a given link configuration G and has two additional properties: it has the largest width W_F and maximum area A_F . A given fault pattern F is said to *cross* the reference fault pattern if $I(F)$ is not contained in $I(RFP)$ and $I(RFP)$ is not contained in $I(F)$.

The following two lemmas are the basis for the necessary and sufficient condition for the existing testing scheme.

Lemma 2.1 [36] Given G , let RFP be the reference fault pattern and F be any fault pattern for G . If F and RFP cross, then F is not catastrophic for G .

Lemma 2.1 provides us with a necessary condition for a fault pattern to be catastrophic. However, not crossing RFP is not sufficient for a fault pattern to be catastrophic.

Lemma 2.2 [36] If F does not satisfy the following property then F is not catastrophic for G : for any column y_i ($0 \leq y_i \leq g_k - 1$) in W and for any link $g \in G = \{g_1, g_2, \dots, g_k\}$

$$x_i \leq \begin{cases} x_{i+g} + 1 & \text{if } i + g \leq g_k - 1 \\ x_i & \text{otherwise} \end{cases}$$

where $j = (i + g) \bmod g_k$.

The above lemmas state necessary conditions for a fault pattern to be catastrophic. We will now show that the combination of the conditions expressed by Lemmas 2.1 and 2.2 constitute a necessary and sufficient condition.

Theorem 2.2 [36] A fault pattern is catastrophic for a link configuration G if and only if

- i) it does not cross the reference fault pattern corresponding to G , and
- ii) it satisfies Lemma 2.2.

The testing algorithm of [36] is constructed from the preceding results. In particular, the algorithm will verify whether the necessary and sufficient conditions expressed by Theorem 2.2 are met.

The algorithm includes a pre-testing phase, ensuring that the width and area of F are not greater than the ones of the RFP. The major steps of the algorithm are: Test for Crossing and test for Property which is Lemma 2.2.

We now give the complete testing algorithm.

Algorithm 2.2: Testing if a given fault pattern F is catastrophic for G

Begin

TEST:=True;

Test for violation of maximal area and width;

if TEST *then*

Test for crossing;

if TEST *then* Test for property;

endif

End.

Test for violation of maximal area and width (Pre-Testing)

Begin

if WRFP < WF or ARFP < AF *then*

TEST:=false

endif

End;

Test for Crossing

Begin

Let $\{x_i\}$ and $\{x_i'\}$ be the row coordinates of F and RFP, respectively.

$i:=0$;

repeat

if $x_i > x_i'$ *then*

TEST:=False

endif

$i:=i+1$;

until $i > g_k$ or not(TEST)

End;

Test for Property

Begin

$i:=0$;

repeat

$j:=1$;

repeat

if $i+g_j \leq g_k-1$ *then* $x_p := x_i + g_j + 1$ *else* $x_p := (x_i + g_j) \bmod g_k$;

if $x_i > x_p$ *then*

TEST:=False

endif

$j:=j+1$;

until $j > k$ or not(TEST)

$i:=i+1$;

until $i > g_k$ or not(TEST)

End;

This testing scheme is proved to have time complexity (kg_k). The scheme is quite efficient. Unfortunately, it has some limitation for which it is not suitable for fault patterns with arbitrary number of faults since it only deals with fault patterns with exactly g_k faults. In practice, it is not likely that fault patterns will always have exactly g_k faults. Therefore, we require a testing scheme that can deal with fault patterns with an arbitrary number of faults.

2.5.3 Existing Testing Scheme 3

In this section, the testing scheme of [48] for a fault pattern to be catastrophic in a linear unidirectional array with k -link redundancy is described.

In this scheme, the array is treated as a connected graph where the processing elements are the nodes while the links are edges in the corresponding graph. All non-faulty processing elements are divided into *chunks*, each of which consists of consecutive non-faulty processing elements. A necessary and sufficient condition for a fault pattern not to be catastrophic is given stating that a fault pattern is not catastrophic if and only if the first chunk (i.e., input of the array) is connected with the last chunk (i.e., output of the array). Based on this necessary and sufficient condition, testing schemes are developed to test the catastrophe of a fault pattern for unidirectional arrays and bidirectional arrays.

The algorithm is organized in the following way. All the chunks are considered in increasing order; from each chunk, excluding the last one, are derived sets of consecutive processing elements (blocks), obtained by extending of one link all the escape paths of the elements of the chunk. The intersection between the current chunk and the previously

constructed blocks will indicate the sets of PEs (new blocks) reachable from the beginning of the array. A chunk is denoted by the pair (x,y) where P_x and P_y are the first and the last PE in the chunk, respectively. Analogously (x',y') denotes the block of PEs $P_{x'} \dots P_{y'}$. A pair (x,y) is minimal in a set X of pairs if, for each (u,v) in X , it holds $x \leq u$, it is maximal if, for each (u,v) in X , it results $u \leq x$. Three possible situations, given below, arise in the comparison between a chunk and a block, and the new block generated:

$$1) x' \leq x \leq y' \leq y$$

$$2) x \leq x' \leq y' \leq y$$

$$3) x \leq x' \leq y \leq y'$$

The algorithm is shown in Figure 2.9.

When all the chunks have been considered if there is no intersection between the last chunk and anyone of the blocks, the fault pattern is catastrophic since no PE outside the fault zone is reachable from the input. The testing algorithm (TEST) is shown on the next page. TEST returns true if and only if F is catastrophic for A .

This testing scheme requires time $O(kn \log k)$, where k is the number of bypass links and n is the number of faults.


```

TEST(F,G)
begin
  S = {(f1-gk+1,f1-1)}; B={(f1-gk+1,f1-1)}
  for i=1,...,n-1 do
    insert (fi+li, fi+1-1) into S
  endfor
  while S ≠ ∅ and B ≠ ∅ do
    let (x',y') be a minimal element of B
    let (x,y) be a minimal element of S
    case
      y' < x      : delete (x',y') from B
      y < x'      : delete (x,y) from S
    otherwise:
      x" := max(x,x')
      for i=1,...,k do
        insert (x"+gi,y+gi) into B
      endfor
      delete (x,y) from S
    endcase
  endwhile
  if in B there is a pair (x',y'), y'+gk > fn+ln
    then return False
  else return True
end

```

Figure 2.10: A testing algorithm

2.6 Chapter Summary

This chapter first gives the background of systolic arrays and techniques achieving fault tolerance in systolic arrays. A known characterization of catastrophic fault patterns in systolic arrays is introduced. Three existing testing schemes are also given.

Chapter 3

Geometric Characterizations of Fault Patterns

3.1 Introduction

In this chapter, fault patterns are geometrically characterized for linear unidirectional arrays, where the number of faults is arbitrary.

A fault pattern can be represented by a Boolean matrix. By examining the Boolean matrix, it is possible to identify several types of geometric objects named after their shapes, i.e., rectangles, staircases and segments.

On one hand, if we look at the entire Boolean matrix, we can identify rectangles which can be combined to form composite objects. On the other hand, if we look at the Boolean matrix one row at a time, we can identify segment type objects. Like in the case of rectangles, it is also possible to combine the segments to form larger segments.

In this chapter, we first represent fault patterns by a Boolean matrix, and then by viewing the matrix in different ways, we identify the geometric objects. The organization of this chapter is as follows: In Section 3.2, a Boolean matrix representation of fault patterns is introduced. Section 3.3 describes three types of geometric objects, and the identifications of geometric objects are given in Section 3.4.

3.2 Boolean Matrix Representation of Fault Patterns

The system being studied is one-dimensional (or linear) arrays of PEs with a given link redundancy with arbitrary number of faults.

Example 3.1 An example of a fault pattern of 5 faults with link redundancy (1,4) is given in Figure 3.1.

Figure 3.1: A fault pattern of 5 faults with link redundancy (1,4)

The fault pattern given in Figure 3.1 is catastrophic since the removal of the faulty PEs and their incident links disconnect the array, as shown in Figure 3.2.

Figure 3.2: A disconnected array of PEs

We now describe the Boolean matrix representation for fault patterns introduced in [34]. This representation will be instrumental in establishing the geometric characterizations for fault patterns and deriving efficient testing algorithms.

Consider a linear array Λ with $\|\Lambda\|$ processing elements, with a link configuration $G=\{g_1, g_2, \dots, g_k\}$ and a fault pattern $F=\{f_1, f_2, \dots, f_n\}$. Without loss of generality, let $\Lambda[1]=f_1$, then W of size $(\lceil \|\Lambda\| / g_k \rceil \times g_k)$ is defined as follows:

$$W[i,j]=A[(i-1)g_k+j] \quad 1 \leq i \leq \lceil \|\Lambda\| / g_k \rceil, 1 \leq j \leq g_k$$

$$W[i,j] = \begin{cases} 1 & \text{if } W[i,j] \text{ is faulty} \\ 0 & \text{if } W[i,j] \text{ is non-faulty} \end{cases}$$

In this representation, the first faulty element $\Lambda[1]$ is mapped onto $W[1,1]$, $\Lambda[g_k]$ is mapped into $W[1,g_k]$, and so on. For every element in Λ , say $\Lambda[r]$, there are two links which connect $\Lambda[r]$ to two other elements in Λ , i.e., one is $\Lambda[r+1]$, the other is $\Lambda[r+g_k]$. Corresponding this fact to the matrix M , for every element in W , say $W[i,j]$, there are two links which connect $W[i,j]$ to two other elements in W , i.e., if $j < g_k$, one is $W[i,j+1]$, the other is $W[i+1,j]$; if $j = g_k$, one is $W[i+1,1]$, the other is $W[i+1,j]$. In general, each element in W , say $W[i,j]$, can 'go right' or 'go down' if $j < g_k$, can go 'wraparound' or 'go down' if $j = g_k$. For simplicity, from now on we will use (i,j) to represent $W[i,j]$.

Example 3.2 The Boolean matrix representation of the fault pattern $F=\{f_1, f_2, f_3, f_4, f_5\}=\{(1,1), (1,4), (2,2), (2,4), (3,3)\}$ is given in Figure 3.3. For example, $(1,1)$ connects to $(1,2)$ and $(2,1)$, $(2,4)$ connects to $(3,1)$ and $(3,4)$.

$$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

**Figure 3.3: A Boolean matrix of a catastrophic fault pattern of 5 faults
with link redundancy {1,4}**

According to the connections in this kind of matrix, we give the definition of a path.

Definition 3.1 A *path* Π from (x_1, y_1) to (x_p, y_p) in a matrix is a set of elements $\Pi = \{(x_1, y_1), (x_2, y_2), \dots, (x_p, y_p)\}$ where there is a link between (x_i, y_i) and (x_{i+1}, y_{i+1}) , $1 \leq i \leq p-1$.

Example 3.3 As an example, a path Π from $(1,2)$ to $(4,1)$ in the matrix representation as shown in Figure 3.3, is given by $\Pi = \{(1,2), (2,2), (2,3), (2,4), (3,1), (4,1)\}$.

3.3 Geometric Objects

3.3.1 Rectangle Type of Geometric Objects

In this section, fault patterns on unidirectional array with 2-link redundancy $G=\{1, g_k\}$ are studied.

Let R be a rectangle whose four vertices are $\{(x_l_R, y_l_R), (x_l_R, y_u_R), (x_u_R, y_l_R), (x_u_R, y_u_R)\}$. It is sufficient to represent R by its lower left and upper right vertices, i.e.: $R=\{(x_l_R, y_l_R), (x_u_R, y_u_R)\}$. We denote $L_x(R), L_y(R), U_x(R), U_y(R)$ as follows:

$$L_x(R)=x_l_R$$

$$L_y(R)=y_l_R$$

$$U_x(R)=x_u_R$$

$$U_y(R)=y_u_R$$

Definition 3.2 A *dead rectangle* is a rectangle R on which each path, originating at the first row and/or column of R and ending at the last row and/or column of R , has at least one faulty element.

For simplicity, we use the notation row/column to express "row and/or column".

Example 3.4 An example of a dead rectangle is given in Figure 3.4. Since each path starting at the first row/column of the rectangle and ending at the last row/column of the rectangle has at least one faulty element, the rectangle is dead.

1	0	0	0	0	1
0	1	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	0	1	0	0	0

Figure 3.4: A dead rectangle

Example 3.5 The rectangle shown in Figure 3.5 is not a dead rectangle since there is a path $\Pi=\{(3,1),(3,2),(4,2),(5,2)\}$, originating at the first column and ending at the last row, which does not have any faulty element.

1	0	0	0	0	1
0	1	0	0	0	1
0	0	0	0	1	0
1	0	0	1	0	0
0	0	1	0	0	0

Figure 3.5: A rectangle is not dead

The following property shows the relationship between a dead rectangle and the number of faults in the rectangle.

Property 3.1 If a rectangle R is a dead rectangle, then the number of faults in R is at least $U_y(R)-L_y(R)+1$, which is the number of columns in R .

Proof: By contradiction let the number of faults in R be less than $U_y(R)-L_y(R)+1$ and R be a dead rectangle. Since the number of columns of R is $U_y(R)-L_y(R)+1$, there must be a column which does not have any fault. This column itself is a path from one element on the first row to one element on the last row and it does not have any fault. This

contradicts the fact that R is a dead rectangle. Therefore, the number of faults in R is at least $U_y(R) - L_y(R) + 1$ $\square$

However, the converse is not true. That is, if the number of faults in R is at least $U_y(R) - L_y(R) + 1$, then R is not necessarily a dead rectangle. For example, in the rectangle R shown in Figure 3.5, the number of faults in R is 8 is greater than $U_y(R) - L_y(R) + 1 = 6$. However, R is not a dead rectangle.

Definition 3.3 A *faulty chain* is a set of faulty elements $C = \{(x_1, y_1), (x_2, y_2), \dots, (x_c, y_c)\}$

where $x_{i+1} = x_i$ or $x_{i+1} = x_i + 1$, and

if $x_{i+1} = x_i$, then $y_{i+1} = y_i + 1$;

if $x_{i+1} = x_i + 1$, then

either $y_{i+1} = y_i$

or $\forall k, 1 \leq k \leq y_i - y_{i+1} - 1, (x_{i+1}, y_{i+1} + k) \in C$.

Example 3.6 An example of a faulty chain $C = \{(1,7), (2,4), (2,5), (2,6), (3,3), (4,3), (5,2)\}$ in the matrix representation is shown in Figure 3.6, and there is another faulty chain $C' = \{(1,.)\}$ as well.

Figure 3.6: Two faulty chains

A faulty chain can form a rectangle R , whose upper right vertex is the first element of the faulty chain and lower left vertex is the last element of the faulty chain.

Example 3.7 As shown in Figure 3.7, two rectangles, one is $\{(5,2),(1,7)\}$, and the other is $\{(1,1),(1,1)\}$, are formed by the faulty chain which is shown in Figure 3.6.

Figure 3.7: Two rectangles formed by two faulty chain

The following property shows the relationship between a fault chain and a dead rectangle.

Property 3.2 Let $C=\{(x_1,y_1),(x_2,y_2),...,(x_c,y_c)\}$ be a faulty chain. Then the rectangle $R=\{(x_c,y_c),(x_1,y_1)\}$ formed by C is a dead rectangle.

Proof: The faulty chain divides the rectangle R into two disjoint parts, and every path from any element in one part to any element in the other part will encounter at least one element on the faulty chain C . Therefore, every path from any element on the first row/column of R to any element on the last row/column of R has at least one faulty element; that is, R is dead. $\square$

Definition 3.4 An element (i,j) in the matrix representation is called a *dead element* if and only if it is either faulty or every path from the first row/column to the last row/column that includes (i,j) also includes a faulty element. An element which is not a dead element is a *live element*.

Example 3.8 Given M in Figure 3.8, elements $(1,1)$, $(1,4)$, $(1,5)$, $(1,6)$, $(2,1)$, $(2,4)$, $(2,5)$, $(2,6)$, $(3,1)$, $(3,2)$, $(3,4)$, $(3,5)$ and $(3,6)$ are dead elements, and the rest are live elements.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Figure 3.8: Dead elements and live elements

Property 3.3 1) Every element in a dead rectangle is a dead element; 2) every element which is not in any dead rectangle is a live element.

Proof:

1) In a dead rectangle, every path from the first row/column to the last row/column has at least one faulty element, and every non-faulty element must be on one of paths from the first row/column to the last row/column. Thus, every non-faulty element in a dead rectangle is a dead element;

2) By contradiction, let some element (i,j) which is not in any dead rectangle be a dead element. If (i,j) is faulty, (i,j) itself is a dead rectangle, contradiction. If (i,j) is non-faulty dead element, every path from the first row/column to the last row/column of the matrix that includes (i,j) also includes a faulty element. Thus, the matrix is a dead rectangle, contradiction. $\square$

Definition 3.5 A *dead row* in M is a row on which all g_k elements are in dead rectangles.

Example 3.9 An example of a dead row is given in Figure 3.9. There are three dead rectangles in this matrix, i.e., $\{(5,1),(1,4)\}$, $\{(5,5),(3,7)\}$ and $\{(5,8),(5,9)\}$. Every element on row(5) is in one of the three dead rectangles; hence, row(5) is a dead row.

1	0	0	1	0	0	0	0	1
0	0	1	0	0	0	0	0	0
0	1	0	0	0	0	1	0	0
1	0	0	0	0	1	0	0	0
1	0	0	0	1	0	0	1	1

Figure 3.9: A dead row

Two rectangles can intersect. It will be interesting to observe the union of two intersecting dead rectangles and the impact on the surrounding area.

Definition 3.6 Given rectangles A and B, if $U_x(B) \leq U_x(A) \leq L_x(B)+1$ and $L_y(B)-1 \leq U_y(A) \leq U_y(B)$, the *concatenation* of A and B (denoted by $C=A \oplus B$) is a rectangle defined by $\{(L_x(A), L_y(A)), (U_x(B), U_y(B))\}$.

Figure 3.10: $C=A \oplus B$

Example 3.10 An example of the concatenation of two intersecting rectangles is shown in Figure 3.10.

The case in which B intersects A at the right bottom vertex of A (i.e., $U_x(B)-1 \leq L_x(A) \leq L_x(B)$ and $L_y(B)-1 \leq U_y(A) \leq U_y(B)$), as shown in Figure 3.11, will be dealt with separately in Section 3.3.2.

Figure 3.11: B intersects A at the right bottom vertex of A

Property 3.4 Let C be the concatenation of A and B, i.e., $C=A \oplus B$. If A and B are dead rectangles, then C is a dead rectangle.

Proof: Every path from the first row/column of C to the last row/column of C must encounter the union of A and B. Since any path entering A must exit A, the path must contain at least one faulty element. Similarly, any path entering B must exit B, and, thus, contain at least one faulty element. Therefore, every path from the first row/column of C to the last row/column of C must contain at least one faulty element. Therefore, C is a dead rectangle. $\square$

Property 3.5 Given dead rectangles A and B, if $L_y(A)=1$, $U_y(B)=g_k$, and $U_x(A) \leq U_x(B) \leq L_x(A)$, then the rectangle $C=\{(L_x(B)+1,1),(U_x(A),U_y(A))\}$ is a dead rectangle, and the rectangle $D=\{(L_x(B),L_y(B)),(U_x(A)-1,g_k)\}$ is a dead rectangle.

Figure 3.12: Referring to Property 3.5

Proof: Let $U_x(A)=i$, $L_x(A)=j$, and $L_x(B)=k$. Consider rectangle $A' = \{(k+1, 1), (j-1, U_y(A))\}$. Since rectangles A and B are dead rectangles, every path coming into A' will contain at least one faulty element. Rectangle A' hence is a dead rectangle. According to Property 3.4, $C = \{(L_x(B)+1, 1), (U_x(A), U_y(A))\}$ is thus a dead rectangle. Consider rectangle $B' = \{(U_x(B)-1, L_y(B)), (i-1, g_k)\}$. Since rectangles A and B are dead rectangles, every path coming out of B' will contain at least one faulty element. Rectangle B' hence is a dead rectangle. According to Property 3.4, $D = \{(L_x(B), L_y(B)), (U_x(A)+1, g_k)\}$ is thus a dead rectangle. $\square$

Example 3.11 The following example shows how to build dead rectangles from a set of faults in a matrix according to the properties of dead rectangle.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 3.13(a): A set of faults

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 3.13(b): Identifying all faulty chains

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Figure 3.13(c): Building dead rectangles according Property 3.2

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Figure 3.13(d): Building dead rectangles according to Properties 3.4 and 3.5

3.3.2 Staircase Type of Geometric Objects

It is possible that two intersecting rectangles do not satisfy the condition of concatenation. Given rectangles A and B, if $U_x(B)-1 \leq L_x(A) \leq L_x(B)$ and $L_y(B)-1 \leq U_y(A) \leq U_y(B)$, then the union of the two rectangles forms a staircase. Unlike the case of concatenation of two dead rectangles, where a new dead rectangle can be derived from the two dead rectangles, A and B cannot form a single bigger dead rectangle. This is

because any element in these two regions, one of which is above B and right to A and the other of which is below A and left to B, can still 'escape' without encountering any part of A and B. This is shown in Figure 3.14. The union of A and B in this case resemble a staircase.

Figure 3.14: An example of a staircase formed by two intersecting rectangles

Definition 3.7 A *staircase* is the union of a set of rectangles $\{R_1, R_2, \dots, R_s\}$, where $U_x(R_{i+1}) - 1 \leq L_x(R_i) \leq L_x(R_{i+1})$ and $L_y(R_{i+1}) - 1 \leq U_y(R_i) \leq U_y(R_{i+1})$, $1 \leq i \leq s-1$.

A staircase S can also be represented as $S = \{R_1, R_2, \dots, R_s\} = \{U(R_1), U(R_2), \dots, U(R_s), L(R_1), L(R_2), \dots, L(R_s)\}$. The set $\{U(R_1), U(R_2), \dots, U(R_s)\}$ is called *upper part* of S , and the set $\{L(R_1), L(R_2), \dots, L(R_s)\}$ is called *lower part* of S , where $U(R_i) = (U_x(R_i), U_y(R_i))$, $L(R_i) = (L_x(R_i), L_y(R_i))$.

Definition 3.8 Given $S = \{R_1, R_2, \dots, R_s\}$, S is a *dead staircase* if $R_1, R_2, \dots, R_s$ are all dead rectangles.

Two staircases can intersect. In the case of rectangles, it is easy to verify, by comparing the coordinates of the rectangles, if they intersect. But, in the case of staircases, the process of determining if two staircases intersect is somewhat involved. Algorithms 3.1 and 3.2 are dedicated to determining if two staircases intersect.

Claim 3.1 The algorithms (i.e., Algorithms 3.1 and 3.2) require $O(m+n)$, where m is the number of vertices of one staircase, n is the number of vertices of the other staircase.

Proof: Since two algorithms scan through vertices of two staircases, the time complexity is $O(m+n)$, where m is the number of vertices of one staircase, n is the number of vertices of the other staircase.

Algorithm 3.1 (*Deciding if two staircases intersect*)

Input: Two staircases.

Output: Telling if these two staircases intersect.

Compare each part (i.e., upper part, lower part) of
one staircase with each part of the other staircase
using Algorithm 3.2;

If (they intersect)

then stop and tell;

else continue;

Given two intersecting staircases, either the union of two staircases might just be another staircase, or the union can look different from a staircase. In the case that the union is not a staircase by itself, it is possible to define a new staircase on this region.

Definition 3.9 Given staircases $S=\{U(R_1), U(R_2), \dots, U(R_m), L(R_1), L(R_2), \dots, L(R_m)\}$, and $S'=\{U(R'_1), U(R'_2), \dots, U(R'_n), L(R'_1), L(R'_2), \dots, L(R'_n)\}$, the *concatenation* of S and S' (denoted by $S''=S \otimes S'$) is a staircase derived from Algorithm 3.3 if S intersects S' .

Algorithm 3.2 (*Deciding if two parts intersect*)

Input: Vertices of two parts, either upper part or lower part.

Output: Telling if the two parts intersect.

(1) Compare the first segment of one part with the first segment of the other part;

If (they do not intersect)

then choose the segment whose y coordinate is smaller as the current segment;

else stop and tell they intersect.

(2) *While* (not the end of anyone of the two parts) {

Compare the current segment with the next segment of another part;

If (they do not intersect)

then choose the segment whose y coordinate is smaller as the current segment;

else stop and tell they intersect.

(3) When the end of one part has been reached, tell they do not intersect.

Algorithm 3.3 (Finding concatenation of two intersecting staircases)

Input: Two intersecting staircases $S = \{U(R_1), U(R_2), \dots, U(R_m), L(R_1), L(R_2), \dots, L(R_m)\}$, $S' = \{U(R'_1), U(R'_2), \dots, U(R'_n), L(R'_1), L(R'_2), \dots, L(R'_n)\}$

Output: A new staircase S'' .

(1) for upper part of S''

$i=1; j=1;$

while ($i \leq m$ and $j \leq n$) {

if ($U_x(R_i) \geq U_x(R'_j)$) and ($U_y(R_i) \geq U_y(R'_j)$))

then {append $U(R'_j)$ into the upper part of S'' ; $j++$;}

if ($U_x(R_i) < U_x(R'_j)$) and ($U_y(R_i) > U_y(R'_j)$))

then {append $U(R_i)$ into the upper part of S'' ; $i++$; $j++$;}

if ($U_x(R_i) \leq U_x(R'_j)$) and ($U_y(R_i) < U_y(R'_j)$))

then {append $U(R_i)$ into the upper part of S'' ; $i++$;}

if ($U_x(R_i) > U_x(R'_j)$) and ($U_y(R_i) < U_y(R'_j)$))

then {append $U(R'_j)$ into the upper part of S'' ; $i++$; $j++$;}

}
(2) for lower part of S''

$i=1; j=1;$

while ($i \leq m$ and $j \leq n$) {

if ($L_x(R_i) \geq L_x(R'_j)$) and ($L_y(R_i) \geq L_y(R'_j)$))

then {append $L(R'_j)$ into the lower part of S'' ; $j++$;}

if ($L_x(R_i) < L_x(R'_j)$) and ($L_y(R_i) > L_y(R'_j)$))

then {append $L(R'_j)$ into the lower part of S'' ; $i++$; $j++$;}

if ($L_x(R_i) \leq L_x(R'_j)$) and ($L_y(R_i) < L_y(R'_j)$))

then {append $L(R_i)$ into the lower part of S'' ; $i++$;}

if ($L_x(R_i) > L_x(R'_j)$) and ($L_y(R_i) < L_y(R'_j)$))

then {append $L(R_i)$ into the lower part of S'' ; $i++$; $j++$;}

}

Example 3.12 An example of Algorithm 3.3 is shown in Figure 3.15.

Figure 3.15: An example of Algorithm 3.3

Claim 3.2 Algorithm 3.3 requires $O(m+n)$ time, where m is the number of vertices of one staircase, n is the number of vertices of the other staircase.

Proof: The time complexity of Algorithm 3.3 is $O(m+n)$, because Algorithm 3.3 scans through vertices of two staircases, where m is the number of vertices of one staircase, n is the number of vertices of the other staircase.

Property 3.6 Given $S'' = S \oplus S'$, S'' is a dead staircase if S and S' are dead staircases.

Proof: Intuitively, there are two general cases for the new staircase derived from two intersecting staircases.

1) S'' is exactly the union of S and S' ; such a case is shown in Figure 3.16(a).

Figure 3.16(a): The new staircase is exactly the union of two intersecting staircases

2) S'' is the union of S and S' plus some new areas. There are three possible locations for the new areas:

2a) The new area is in the starting area of S and S' , as shown in Figure 3.16(b). By Algorithm 3.3, $U(R_1)$ of S will be taken as $U(R_1)$ of S'' , and $L(R_1)$ of S' will be taken as $L(R_1)$ of S'' , so that the shaded area shown in Figure 3.17(b) will inevitably be included in S'' . Every path coming out of this new area will come into S or S' , and S and S' consist of dead rectangles. Consequently, every path coming out of the new area will have at least one faulty element. Thus, the new area which itself is a staircase is a dead staircase.

Figure 3.16(b): New area is at the starting area of S and S'

2b) The new area is in the middle area of S and S' , and is totally surrounded by S and S' , as shown in Figure 3.16(c). The number of this kind of new area in S'' is arbitrary, could be 0, or 1, or more than 1. Every path coming into of this kind of new area are from S and S' , which are dead staircases. As a result, every path coming into the new area has at least one faulty element. Thus the new area which itself is a staircase is a dead staircase.

Figure 3.16(c): New area is in the middle area of S and S'

2c) The new area is in the ending area of S and S', as shown in Figure 3.16(d). For the same reason as 2b), every path coming into this new area has at least one faulty element, resulting in the new area to be a dead staircase.

Figure 3.16(d): The new area is in the ending area of S and S'

General speaking, S'' is either exactly the union of dead staircases S and S', or the union of dead staircases S and S' plus some new areas which are dead staircases. Therefore, S'' is a dead staircase. $\square$

Example 3.13 Section 3.3.1 introduces rectangle type of geometric objects, while this section brings up staircase type of geometric objects. Given a matrix representation of a fault pattern, it will be interesting to see how these geometric objects look like in the

matrix. The following example will illustrate the process of building geometric objects from a set of fault elements in the matrix.

1	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0
1	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0
1	0	0	1	0	1	1	0	1	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	1	1	0	1	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0	1	0	0	0	1	0	0	0
0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0

Figure 3.17(a): Starting from a set of faults in a matrix

1	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0
1	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	1	1	0	1	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0	1	0	0	0	1	0	0	0
0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0

Figure 3.17(b): Identifying all faulty chains according to the definition of faulty chain

Figure 3.17(c): Identifying dead rectangles from faulty chains
according to Property 3.2

Figure 3.17(d): Expanding dead rectangles and identify
all dead staircases according to Priorities 3.4,
3.5 and the definition of dead staircase

3.3.3 Segment Type of Geometric Objects

Sections 3.3.1 and 3.3.2 study the fault patterns with respect to 2-link redundancy by studying geometric objects (i.e., rectangles, staircases). Since rectangles and staircases span multiple rows, possibly the whole matrix, we study the fault patterns based on the global knowledge of the fault patterns. In this section, a different approach is used to study the more general case where link redundancy $G=\{g_1, g_2, \dots, g_k\}$. The approach used in this section is somewhat "local" approach; that is, the geometric objects we are studying are on the single rows, so that we study fault patterns based on the local knowledge of the fault patterns, i.e., the knowledge of the current row and the previous row.

Definition 3.10 A *dead element* is

(1) a faulty element

(2) a non-faulty element (x, y) , if

$\exists i$, such that $y - g_i \geq 1$, $(x, y - g_1)$, ..., $(x, y - g_i)$, $(x - 1, y - g_{i+1} + g_k)$, $(x - 1, y - g_{i+2} + g_k)$, ..., $(x - 1, y)$ are all dead elements;

or $(x - 1, y - g_1 + g_k)$, $(x - 1, y - g_2 + g_k)$, ..., $(x - 1, y)$ are all dead elements.

A *live element* is an element which is not a dead element.

Example 3.14 In the matrix shown on Figure 3.18 where $G=\{1, 3, 5, 7\}$, elements $(1, 1), (1, 3), (1, 5), (1, 7), (2, 2), (2, 4)$ and $(2, 6)$ are dead element since they are faulty. Element $(2, 1)$ is a dead element since elements $(1, 7)$, $(1, 5)$, $(1, 3)$ and $(1, 1)$ are dead elements. Element $(2, 3)$ is a dead element since elements $(2, 2)$, $(1, 7), (1, 5)$ and $(1, 3)$ are all dead elements. For the same reason $(2, 5)$, $(2, 7)$ and all elements on the third row are dead

elements. Any element other than dead elements is a live element, i.e., elements (1,2), (1,4) and (1,6).

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 3.18: Examples of dead elements in a matrix where $G=\{1,3,5,7\}$

Definition 3.11 A *dead segment* is

- (1) a set of dead elements $S=\{(x,y),(x,y+1),\dots,(x,y+d)\}$, where $0 \leq d \leq g_k-1$; or
- (2) if $S_1=\{(x,y),\dots,(x,y+d)\}$ and $S_2=\{(x,y'),\dots,(x,y'+h)\}$ are dead segments and if $y'=y+d+1$, then $S=\{(x,y),\dots,(x,y+h')\}$ is a *dead segment*; or
- (3) if $S_1=\{(x,y),\dots,(x,g_k)\}$ and $S_2=\{(x+1,1),\dots,(x+1,d)\}$ are dead segments where $1 \leq d \leq g_k$, then $S=\{(x,y),\dots,(x,g_k),(x+1,1),\dots,(x+1,d)\}$ is a *dead segment*.

Example 3.15 The following example shows how to derive dead segments from a set of faulty elements in a matrix. The link redundancy in this example is $G=\{1,4,7\}$.

$$\begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Figure 3.19(a): A set of faulty elements in a matrix

$$\begin{bmatrix}
 \overline{1} & \overline{1} & 0 & \overline{1} & 0 & 0 & \overline{1} \\
 0 & 1 & 0 & 0 & 1 & 1 & 0 \\
 0 & 1 & 0 & 1 & 0 & 0 & 1 \\
 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
 1 & 0 & 1 & 0 & 0 & 1 & 0 \\
 0 & 1 & 0 & 0 & 0 & 1 & 0
 \end{bmatrix}$$

Figure 3.19(b): Identifying dead segments on the first row
which are faulty elements

$$\begin{bmatrix}
 \overline{1} & \overline{1} & 0 & \overline{1} & 0 & 0 & \overline{1} \\
 \overline{0} & \overline{1} & 0 & 0 & \overline{1} & \overline{1} & 0 \\
 0 & \overline{1} & 0 & \overline{1} & 0 & 0 & \overline{1} \\
 0 & 0 & \overline{1} & 0 & 0 & \overline{1} & 0 \\
 \overline{1} & 0 & \overline{1} & \overline{0} & 0 & \overline{1} & \overline{0} \\
 \overline{0} & \overline{1} & \overline{0} & \overline{0} & 0 & \overline{1} & \overline{0}
 \end{bmatrix}$$

Figure 3.19(c): Identifying dead segments on each row
according to Definition 3.11

3.4 Chapter Summary

In this chapter, fault patterns for unidirectional linear arrays are geometrically characterized. Three types of geometric objects (i.e., rectangles, staircases and segments) are studied and properties of these geometric objects are given. Based on these geometric characterizations, testing schemes to test whether a fault pattern is catastrophic will be given in Chapter 4 and Chapter 5.

Chapter 4

A Testing Scheme Using Rectangle and Staircase Objects

4.1 Introduction

In this chapter, fault patterns which are catastrophic for unidirectional linear arrays are studied using rectangle and staircase objects. Most of the previous work on testing scheme determines if a fault pattern is catastrophic only when the number of faults in the fault pattern is exactly g_k , that is the length of the longest bypass link. The problem presentation and the solution techniques are not easily extendible to the general case where the number of faults is arbitrary. In this chapter, based on the geometric characterization for fault patterns given in Chapter 3, a necessary and sufficient condition is established to determine if a fault pattern is catastrophic with respect to 2-link redundancy. Based on the necessary and sufficient condition, a testing algorithm whose complexity is $O(n^3)$ is developed .

The organization of this chapter is as follows. The previous work on testing scheme which determines if a fault pattern is catastrophic is presented in Section 4.2. In Section 4.3, a necessary and sufficient condition for a fault pattern to be catastrophic for arbitrary number of faults is given, then, a complete testing algorithm is presented to determine if a given fault pattern is catastrophic. The worst time complexity of this testing algorithm is $O(n^3)$, where n is the number of faults.

4.2 A Testing Scheme

The new testing scheme that is described in this section has the advantage over the existing testing scheme in that it can be used to test fault patterns with arbitrary number of faults. This scheme is based on the geometric characterization given in Chapter 3. Recall from Chapter 3, that the characterization identifies several geometric objects, such as rectangles and staircases. Based on the characteristics of these objects, if it is possible to determine if a fault pattern is catastrophic.

In the following we will describe the basis for the testing algorithm by deriving a necessary and sufficient condition for fault patterns to be catastrophic. Once this condition is derived, the development of the algorithm will be straightforward.

4.2.1 A Necessary and Sufficient Condition

Recall from Chapter 3, that the definition of a *path* is a set of elements where there is a link between each two consecutive ones. Here we give the definition of a *live path*.

Definition 4.1 A *live path* is a path on which every element is live. Given two live elements x and y , y is *reachable* from x if there exists a live path from x to y .

In the following, we give the definition of entry points, then claim that the set of entry points for each row is actually the set of live elements for that row.

Definition 4.2 The set of *entry points* of row(*i*), denoted by E_i , is given by the following recursive definition:

$$E_i = \{\text{live elements in row } (i) \text{ reachable from some element in } E_{i-1}\} \text{ if } i > 1, \text{ and} \\ E_1 = \{\text{live elements in row } (1)\}$$

Claim 4.1 Let L_i be the set of live elements of row (*i*), then $E_i = L_i$.

Proof: By induction on *i*.

By definition of E_1 , the claim trivially holds for $i=1$.

Let it hold for E_i , $i > 1$. Consider row($i+1$). By contradiction, define the set of live elements which are not entry points $W_{i+1} = L_{i+1} - E_{i+1} \neq \emptyset$. Let *j* be the smallest index such that $(i+1, j) \in W_{i+1}$. Depending on whether $j=1$ or $j > 1$, we shall consider two cases.

Case 1 $j=1$: Since $(i+1, 1) \notin E_{i+1}$, then $(i, 1)$ is not in E_i , (which is L_i by inductive hypothesis). Then, by Property 3.5, $(i+1, j)$ is dead. A clear contradiction.

Case 2 $j > 1$: Since $x = (i+1, j) \notin E_{i+1}$, then (i, j) is dead; also $y = (i+1, j-1) \in L_{i+1}$ (otherwise, by Property 3.4, $(i+1, j)$ is dead). Now, *y* cannot be in E_{i+1} (otherwise, $x \in E_{i+1}$). This contradicts the fact that *j* is the smallest index for which $(i+1, j) \in W_{i+1}$.

Therefore, the claim holds. $\square$

A necessary and sufficient condition, for a fault pattern of arbitrary number of faulty elements to be catastrophic for a 2-link unidirectional systolic array, is the following.

Theorem 4.1 A fault pattern is catastrophic *if and only if* it has at least one dead row.

Proof:

(*if part*) By contradiction let the fault pattern be catastrophic, and $\forall i \ L_i \neq \emptyset$. Consider the element sequence $l_1, l_2, \dots, l_{\text{Last}}$, $l_i \in L_i$, $l_i = (x_i, y_i)$, where l_{Last} is an arbitrary element in $L_{\text{Last}} \neq \emptyset$; and l_i , $i < \text{Last}$, is recursively constructed from l_{i+1} as follows: L_{i+1} (which by

Claim 4.1 is E_{i+1} is the set of live elements in $\text{row}(i+1)$ reachable from some live element in $\text{row}(i)$. Thus for every $a \in L_{i+1}$, there exists at least an element $b \in L_i$ such that a is reachable from b . Choose l_i be the element in L_i from which l_{i+1} is reachable. Therefore, the sequence of $l_1, l_2, \dots, l_{\text{Last}}$, defines a path of live elements, l_{Last} is reachable from l_1 ; thus contradicts the fact that the fault pattern is catastrophic.

(only if part) By contradiction, let $L_j = \emptyset$ and the fault pattern be not catastrophic. Since the fault pattern is not catastrophic, then $\exists l_1, l_2, \dots, l_{\text{Last}}$ such that $l_i \in L_i$ and $l_i \in \text{row}(i)$. Therefore, $\forall i L_i = E_i \neq \emptyset$. Contradicting $L_j = \emptyset$. $\square$

4.2.2 A Testing Algorithm

Theorem 4.1 establishes a necessary and sufficient condition for a fault pattern to be catastrophic; based on it, we are able to develop an algorithm to test if a given fault pattern is catastrophic. To prove that a given fault pattern is catastrophic, it is sufficient to show that there exists a dead row. Recall from Definition 3.5, a dead row is a row on which all g_k elements are in dead rectangles. As a result, the task of testing if a fault pattern is catastrophic becomes a task of building dead rectangles, and checking if any dead rectangle covers every column of M . In the process of building dead rectangles, staircases may be built. Hence, the main focus of the testing algorithm is not only dead rectangles, but also dead staircases.

Instead of identifying all the dead geometric objects and then checking if the fault pattern is catastrophic, we will construct dead geometric objects and check for catastrophe simultaneously. During the process of constructing dead rectangles and

staircases, if a dead row is encountered, the process can stop and indicate that the fault pattern is already catastrophic. The algorithm is based on this principle.

The complete testing algorithm is presented as follows.

Algorithm 4.1

Outline: This testing algorithm takes a fault pattern, checks if the number of faults is less than g_k ; if so, it reports that the fault pattern is not catastrophic. Otherwise, the algorithm maps the fault pattern onto a Boolean matrix. Based on Property 3.4 and Property 3.5, the algorithm will check each two faults to see if the two faults can form a dead rectangle. If so, the newly formed dead rectangle will be appended to the set of faults, which will in turn be checked with all other faults and/or dead rectangles to see if a bigger dead geometric object can be formed. If the condition of forming a staircase is met, the newly formed staircase will also be appended to the set of faults, which will also in turn be checked with all other faults, dead rectangles and/or dead staircases to see if a larger dead geometric object can be formed. During this process, if a dead row is encountered, the testing algorithm will stop and report the given fault pattern is catastrophic. If the end of the set of dead geometric objects is reached, the algorithm will report it is not catastrophic.

Algorithm 4.1

Input: A set of faults $F=\{f_1, f_2, \dots, f_n\}$ and link redundancy $G=\{1, g_k\}$.

Output: Report whether or not the fault pattern is catastrophic.

Step 1: *If* ($n < g_k$)
 then stop and report it is not catastrophic;
 else continue;

Step 2: Map F to a Boolean matrix.

Step 3: *while* (not the last fault in F) {

 starting from the second fault, compare each fault with each
 previous fault to form a dead rectangle if the condition of forming
 a dead rectangle is met (i.e., Property 3.4 or Property 3.5), append
 this newly formed dead rectangle at the end of F ;

}

Step 4: *while* (not the end of F) {

 starting from the first dead rectangle in F which is the next
 element to the last fault in F , check each dead rectangle
 (or dead staircase) with each previous fault (or dead rectangle
 or dead staircase) to see if they intersect by means of Algorithm 3.1;
 if they intersect, then form a new dead staircase by means of
 Algorithm 3.3. The new dead staircase can be represented by
 $S=\{U(R_1), U(R_2), \dots, U(R_s), L(R_1), L(R_2), \dots, L(R_s)\}$. Then check
 if there is a dead row within S by checking if the following condition
 is met: $U_x(R_s) \wedge L_x(R_1)$ and $L_y(R_1)=1$ and $U_y(R_s)=g_k$

if this is the case
 then stop and report it is a catastrophic fault pattern;
 else continue.

}

if the end of F is reached,
 then report this is not a catastrophic fault pattern.

The following are two examples of testing whether or not a fault pattern is catastrophic.

Example 4.1 Consider the Boolean matrix of a fault pattern given in Figure 4.1(a), whose link redundancy $G=\{1,8\}$. The number of faults in this fault pattern is 14 which is greater than $g_k=8$.

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 4.1(a): Boolean matrix of a fault pattern

Two dead rectangles, $\{(2,1), (1,2)\}$ and $\{(8,8), (8,8)\}$, are formed, as shown in Figure 4.1(b).

$$\begin{bmatrix} \boxed{1} & 0 & 0 & 0 & 0 & 0 & 0 & \boxed{1} \\ \bullet & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 4.1(b): Intermediate result of Algorithm 4.1

As shown in Figure 4.1(c), two larger dead rectangles, $\{(3,1), (1,3)\}$ and $\{(3,7), (1,8)\}$, are built.

$$\begin{bmatrix} \boxed{\begin{matrix} 1 & 1 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{matrix}} & 0 & 0 & 0 & \boxed{\begin{matrix} 0 & 1 \\ 0 & 0 \\ 1 & 1 \end{matrix}} \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 4.1(c): Intermediate result of Algorithm 4.1

Figure 4.1(d) shows that two larger dead rectangles, $\{(4,1), (1,3)\}$ and $\{(4,6), (1,8)\}$, are constructed.

$$\begin{bmatrix} \boxed{\begin{matrix} 1 & 1 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{matrix}} & 0 & 0 & \boxed{\begin{matrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{matrix}} \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 4.1(d): Intermediate result of Algorithm 4.1

$$\begin{bmatrix} \boxed{\begin{matrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \end{matrix}} \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 4.1(e): The algorithm stops when a dead row is found

As shown in Figure 4.1(e), a single larger dead rectangle, $\{(5,1), (1,8)\}$, is formed. The algorithm stops since there is at least a dead row in the dead rectangle, which leads to the conclusion that the fault pattern is catastrophic.

Example 4.2 Consider the Boolean matrix of a fault pattern given in Figure 4.2(a), whose link redundancy $G=\{1,8\}$. The number of faults in this fault pattern is 14 which is greater than $g_k=13$.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Figure 4.2(a): Boolean matrix of a fault pattern

Three dead rectangles, $\{(1,1), (1,1)\}$, $\{(1,4), (1,4)\}$ and $\{(2,5), (2,5)\}$, are identified, as shown in Figure 4.2(b).

$$\begin{bmatrix} \boxed{1} & 0 & 0 & 0 & \boxed{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \boxed{0} & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Figure 4.2(b): Intermediate result of Algorithm 4.1

As shown in Figure 4.2(c), two more dead rectangles $\{(3,5), (2,6)\}$ and $\{(3,7), (3,8)\}$ are built.

1	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	1	0	1	1
0	0	0	0	1	0	0	0
0	0	1	0	0	0	0	0
0	1	0	0	0	1	0	0
1	0	0	0	0	0	1	0

Figure 4.2(c): Intermediate result of Algorithm 4.1

Four more dead rectangles, $\{(6,2), (5,3)\}$, $\{(6,6), (6,6)\}$, $\{(7,1), (6,2)\}$ and $\{(7,7), (7,7)\}$ are formed, as shown in Figure 4.2(d).

1	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	1	0	1	1
0	0	0	0	1	0	0	0
0	0	1	0	0	0	0	0
0	1	0	0	0	0	0	0
1	0	0	0	0	0	1	0

Figure 4.2(d): Intermediate result of Algorithm 4.1

1	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	1	0	1	1
0	0	0	0	1	0	0	0
0	0	1	0	0	0	0	0
0	1	0	0	0	0	0	0
1	0	0	0	0	0	1	0

Figure 4.2(e): No dead row is found

Finally as shown on Figure 4.2(e), all dead geometric objects are: two dead rectangles, $\{(1,1), (1,1)\}$ and $\{(7,1), (5,3)\}$; and two dead staircases, $\{(1,5), (2,6), (3,8), (1,5), (2,5), (4,5)\}$ and $\{(6,6), (7,7), (6,6), (7,7)\}$.

Since no dead row is found when all possible dead rectangles and staircases are built, the given fault pattern is not catastrophic. Consequently, there must exist escape path(s) in this fault pattern. The escape path here means a path in the fault pattern from some non-faulty element on the first row to some non-faulty element on the last row, on which every element is non-faulty. An example of such an escape path is $\{(1,8), (2,8), (3,1), (3,2), (3,3), (3,4), (4,4), (5,4), (6,4), (7,4)\}$.

Claim 4.2 Algorithm 4.1 requires $O(n^3)$ time, where n is the number of faults.

Proof: Initially the number of elements of F is n . Mapping F to a Boolean matrix needs $O(n)$. Then the algorithm checks each fault with all other faults to see if a dead rectangle can be formed. This checking needs $O(n)$. The number of dead rectangles and dead staircases generated from fault elements is at most n . Hence, The number of elements in F is $O(n)$. Then the algorithm check each dead rectangle or dead staircase with all other elements in F . To check if two elements (i.e., dead rectangles or dead staircases) intersect requires $O(n)$ according to Claim 3.1. To form a new dead staircase from two intersecting dead staircases requires $O(n)$. Each element of F (fault, dead rectangle or dead staircase) checks with $O(n)$ elements. Therefore, the algorithm requires $O(n^3)$ time. $\square$

4.3 Alternative Approach

An alternative approach derives from considering dead rectangles and staircases as obstacles; the problem of testing for catastrophe becomes the problem of determining the existence of an obstacle-avoiding path. Specifically, we can consider the array as a two dimensional grid with wrap-around links; that is, point (i, g_k) is connected to point $(i+1, 1)$. In this grid, movement is unidirectional; that is, from point (x, y) , only $(x+1, y)$ and $(x, y+1)$ can be reached if $y < g_k$, and only $(x+1, y)$ and $(x+1, 1)$ can be reached if $y = g_k$. In this grid, there are $O(n)$ rectilinear obstacles which are just rectangles, possibly intersecting. Then the problem is: given two points (a, b) and (c, d) in the grid, to determine if there exists an obstacle-avoiding unidirectional path from (a, b) to (c, d) .

The problem of finding an obstacle-avoiding path in a two dimensional grid have been dealt with in [3, 4, 5, 6]. Particularly, [3] and [4] deal with the parallel complexity of shortest path computation on "grid" graphs (weighted edges - so it captures the "obstacle" situation of infinite weight, but they do not have any wrap-around edges). In [63], it is shown that preprocessing a planar s-t graph can be done so as to perform path queries (yes/no) very fast. Unfortunately our graph is not planar because of the wrap-around edges. Notice that all of the above approaches do not allow for the wrap-around links.

All these approaches lead to $O(n \log n)$ bounds in absence of wrap-around links. It might be possible to extend the above approach to solve our problem with the same complexity. However, to be competitive, any such extended techniques must improve upon the $O(n)$ bound proved in the next chapter.

4.4 Chapter Summary

In this chapter, the problem of determining a fault pattern to be catastrophic for unidirectional linear array with 2-link redundancy has been studied. We study the general case in which the number of faults is arbitrary by establishing a necessary and sufficient condition and deriving a testing algorithm, both of which determine a fault pattern to be catastrophic. An alternative approach is also discussed.

Chapter 5

An Efficient Testing Scheme for Using Segment Objects

5.1 Introduction

In this chapter, fault patterns which are catastrophic for unidirectional linear arrays with arbitrary link redundancy are studied. Following a different approach based on the geometric characterization for fault patterns given in Chapter 3, a necessary and sufficient condition is established for a fault pattern with arbitrary number of faults to be catastrophic for arbitrary link redundancy. Based on the necessary and sufficient condition, an efficient testing algorithm with $O(kn)$ is developed, where k is the number of bypass links and n is the number of faults.

The organization of this chapter is as follows. In Section 5.2 a necessary and sufficient condition for a fault pattern to be catastrophic is presented. The efficient testing algorithm based on this necessary and sufficient condition is given in Section 5.3.

5.2 A Necessary and Sufficient Condition

In the following, we establish a necessary and sufficient condition for a fault pattern, with arbitrary number of faults for an arbitrary link redundancy, to be catastrophic. This condition is based on the geometric characterization given in Chapter 3. Based on this necessary and sufficient condition, an efficient testing scheme will be presented in Section 5.3.

The definitions of a *live path* and *entry points* given in Chapter 4 were only for fault patterns with 2-link redundancy. In case of k -link redundancy, we slightly modify the definitions as follows.

Definition 5.1 A *live path* is a path on which every element is live. Given live elements x and y , y is *reachable* from x if there exists a live path from x to y .

Definition 5.2 The set of *entry points* of row (i) is:

$E_i = \{\text{live elements in row } (i) \text{ reachable from some element in } E_{i-1}\}$ if $i > 1$, and

$E_1 = \{\text{live elements in row } (1)\}$

As for the case $k=2$ considered in Chapter 4, we claim that for every row the set of entry points coincides with the set of live elements, also for an arbitrary k .

Claim 5.1 Let L_i be the set of live elements of row (i) , then $E_i = L_i$.

Proof: By induction on i .

By definition of E_1 , the claim trivially holds for $i=1$.

Let it hold for E_i , $i > 1$. Consider $\text{row}(i+1)$. By contradiction, define the set of live elements which are not entry points $W_{i+1} = L_{i+1} - E_{i+1} \neq \emptyset$. Let j be the smallest index such that $(i+1, j) \in W_{i+1}$. Depending on whether $j=1$ or $j>1$, we shall consider two cases.

Case 1 $j=1$: Since $x=(i+1, 1) \notin E_{i+1}$, then all $(i, g_k), (i, 1-g_2+g_k), \dots, (i, 1)$ are not in E_i , (which is L_i by inductive hypothesis). Then, by Definition 3.11-(2), $x=(i+1, j)$ is dead. A clear contradiction.

Figure 5.1: Claim 5.1 proof, Case 1, $j=1$

Case 2 $j>1$: Since $x=(i+1, j) \notin E_{i+1}$, then $\exists l$ such that all $(i, j-g_k+g_k), (i, j-g_{k-1}+g_k), \dots, (i, j-g_1+g_k)$ are not entry points and by inductive hypothesis are dead. Since $x=(i+1, j)$ is live, then $\exists d, 1 \leq d \leq l-1$ such that $y=(i+1, j-g_d) \in L_{i+1}$, otherwise, by Definition 3.11-(2), $x=(i+1, j)$ is dead. Now, y cannot be in E_{i+1} (otherwise, $x \in E_{i+1}$). This contradicts the fact that j is the smallest index for which $(i+1, j) \in W_{i+1}$.

Figure 5.2: Claim 5.1 proof, Case 2, $j>1$.

Therefore, the claim holds. $\square$

Based on this claim and the characterization given in Chapter 3, a necessary and sufficient condition, for a fault pattern of arbitrary number of faulty elements to be catastrophic for a k -link unidirectional linear array, will be presented below.

Theorem 5.1 A fault pattern is catastrophic *if and only if* it has at least one dead segment of g_k elements.

Proof:

(if part) By contradiction let the fault pattern be catastrophic, and $\forall i, L_i \neq \emptyset$. Consider the element sequence $l_1, l_2, \dots, l_{Last}$, $l_i \in L_i$, $l_i = (x_i, y_i)$, where l_{Last} is an arbitrary element in $L_{Last} \neq \emptyset$; and l_i , $i < Last$, is recursively constructed from l_{i+1} as follows: L_{i+1} (which by Claim 5.1 is E_{i+1}) is the set of live elements in row($i+1$) reachable from some live element in row(i). Thus for every $a \in L_{i+1}$, there exists at least an element $b \in L_i$ such that a is reachable from b . Choose l_i be the element in L_i from which l_{i+1} is reachable. Therefore, the sequence of $l_1, l_2, \dots, l_{Last}$, defines a path of live elements, l_{Last} is reachable from l_1 ; thus contradicts the fact that the fault pattern is catastrophic.

(only if part) By contradiction, let $L_j = \emptyset$ and the fault pattern be not catastrophic. Since the fault pattern is not catastrophic, then $\exists l_1, l_2, \dots, l_{Last}$ such that $l_i \in L_i$ and $l_i \in \text{row}(i)$. Contradicting $L_j = \emptyset$. $\square$

5.3 A Testing Algorithm

According to Theorem 5.1, whether a faulty pattern is catastrophic depends only on the maximum number of elements in the dead segments. Thus, a testing algorithm can just construct the dead segments; the pattern is catastrophic if and only if a dead segment of g_k elements is detected. The testing can be done while constructing the set of dead segments; the construction will stop when either a dead segment of size g_k is found or all dead segments have been constructed and checked.

5.3.1 Gaps and Shadows

It is possible for the Boolean matrix to have several contiguous rows (called gap) which do not contain any faulty elements. Should this be the case, the following theorem shows that the given fault pattern can be partitioned into smaller fault patterns such that,

1. the Boolean matrix of each of these fault patterns does not have any gap;
2. the original fault pattern is catastrophic if and only if at least one of these smaller fault patterns is catastrophic.

Theorem 5.2 Given the Boolean matrix representation of a fault pattern $F = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ $\exists i$ such that $x_{i+1} - x_i > 2$, let $F_1 = \{(x_1, y_1), (x_2, y_2), \dots, (x_i, y_i)\}$ and $F_2 = \{(x_{i+1}, y_{i+1}), (x_{i+2}, y_{i+2}), \dots, (x_n, y_n)\}$; then F is catastrophic *if and only if* F_1 or F_2 is catastrophic.

Proof:

(*if part*) If F_1 or F_2 is catastrophic, then obviously F is catastrophic.

(Only if part) By contradiction, let F_1 and F_2 be not catastrophic. Then there exists a live path L_1 in F_1 from row(1) to row(x_i), and a live path L_2 in F_2 from row(x_{i+1}) to the last row of F . Since row(x_i+1) does not have any faulty element, then (by Definitions 3.10 and 3.11), the only possible dead segment in this row has the form $\{(x_i+1,1), \dots, (x_i+1,j)\}$ where $j \geq 1$, (see Figures 5.3 and 5.4), and the rest of the row elements are live. Thus, there is no dead segment in row (x_i+2). Therefore L_2 can always be connected with L_1 . In other words, there exists a live path from the first row of F to the last row of F , contradicting the assumption that F is catastrophic. $\square$

Figure 5.3: Theorem 5.2

Figure 5.4: Theorem 5.2 proof

Example 5.1 The following is an example illustrating Theorem 5.2. In the fault pattern shown in Figure 5.5 where $G=(1,3,7)$, $F=\{(1,1), (1,4), (1,7), (2,1), (2,5), (2,6), (2,7), (5,1), (5,2), (5,3), (5,4), (5,5), (5,6), (5,7)\}$. In this fault pattern, $5-2 \geq 2$. Thus, let $F_1=\{(1,1), (1,4), (1,7), (2,1), (2,5), (2,6), (2,7)\}$ and $F_2=\{(5,1), (5,2), (5,3), (5,4), (5,5), (5,6), (5,7)\}$. Obviously F_2 is a catastrophic fault pattern. Therefore F is catastrophic.

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Figure 5.5: An example of Theorem 5.2.

Definition 5.3 Let S be a segment, S' be a segment at distance d from S if

- 1) $|S| = |S'|$; and
- 2) $\forall (x,y) \in S$ either $(x,y-d) \in S'$ or $(x-1,y-d+g_k) \in S'$.

S is called the *shadow* of S' at distance d .

Given a segment S , let $x(S)$ be the x coordinate of the first element of S , $y_F(S)$ be the y coordinate of the first element of S , $y_L(S)$ be the y coordinate of the last element of S .

Definition 5.4 Let Ω_i, Ω_j be two segments, $\Omega_i \triangleleft \Omega_j$

if $y_L(\Omega_i) < y_F(\Omega_j)$ and $x(\Omega_i) = x(\Omega_j)$, or $x(\Omega_i) < x(\Omega_j)$.

Lemma 5.1 Let S be a segment and S_{g_i} be the segment at distance g_i from S , ($1 \leq i \leq k$).

The dead segments in S are those in the set

$$\{\text{faulty elements in } S\} \cup \left\{ \bigcap_{i=1}^k \text{shadows of dead segments of } S_{g_i} \text{ at distance } g_i \right\}$$

Proof: According to the definition of dead element, the lemma trivially holds.

Definition 5.5 Let $DS[i]$ be the set of the dead segments of $row(i)$, $DS[j]$ be the set of the dead segments of $row(j)$, the *concatenation* of $DS[i]$ and $DS[j]$, denoted as $DS[i] \bullet DS[j]$, is the union of $DS[i]$ and $DS[j]$.

5.3.2 The Algorithm

The following is a complete testing algorithm which determines if an arbitrary fault pattern is catastrophic for unidirectional linear arrays with arbitrary k -link redundancy.

Informally, the algorithm proceeds as follows. The given fault pattern F is decomposed into patterns which do not contain any gap. By Theorem 5.2, F is catastrophic if and only if at least one of the pattern is catastrophic. By Theorem 5.1, any such pattern F' is catastrophic if and only if there exists a dead segment of size g_k . Thus, the algorithm constructs all the dead segments; in particular, it constructs $DS[i]$ (the set of dead segments of $row(i)$), given $DS[i-1]$ (the set of dead segments of $row(i-1)$). By Definition 3.11, a segment S is dead if all k segments at distance $g_1, g_2, \dots, g_k$ from S are dead. The algorithm will consider only those segments which are shadows at distance g_k from the segments in $DS[i-1]$. In fact, any element of $row(i)$, which is the shadow at distance g_k of a live element, is live; in other words, the only possible candidates for dead segments are the ones considered. Let S be a shadow at distance g_k from some segment in $DS[i-1]$. The algorithm determines which parts of the segments in $DS[i-1] \bullet DS[i]$ are at distance $g_1, g_2, \dots, g_{k-1}$ from S and compute the intersection of all these parts. Any element in S is dead if and only if it belongs to this intersection. If, at any time, a dead

segment of size g_k is encountered, the fault pattern is catastrophic. If all dead segments have been constructed and no dead segment of size g_k has been found, the pattern is not catastrophic.

Algorithm 5.1

Input: A Boolean matrix representation of a fault pattern F .

Output: Report whether or not F is catastrophic.

The algorithm first checks if F is empty. If it is empty, the algorithm stops and reports that the fault pattern is not catastrophic; otherwise the algorithm scans linearly the fault pattern to detect the existence of gaps (i.e., if $x_{i+1}-x_i>2$). When the first gap is found, it decomposes F into two fault patterns F_1 and F_2 such that F_1 does not contain any gap. Then the algorithm calls a procedure (DeadSegment) to check if F_1 is catastrophic. If F_1 is catastrophic, the algorithm stops; otherwise, it recursively call itself with F_2 as its input. Note that if F does not contain any gap, then $F_1=F$ and $F_2=\emptyset$. The flow chart of the algorithm is shown in Figure 5.6.

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No. 2)

Figure 5.6: The flow chart of Algorithm 5.1

DeadSegment(F)

Input: A Boolean representation of F which does not contain any gaps.

Output: TRUE if F is catastrophic; FALSE if F is not catastrophic.

Structure Used:

$DS[i]$ is an ordered (by the (x,y) coordinates of the starting element) set of dead segments of row(i), and it is implemented as a linked list. For simplicity, let $DS[i](k)$ denote the k th dead segment in $DS[i]$.

$DS[i] \bullet DS[j]$ is the concatenation of $DS[i]$ and $DS[j]$, and it is also implemented as a linked list. For simplicity, let $(DS[i] \bullet DS[j])(k)$ denote the k th dead segment in the concatenation of $DS[i]$ and $DS[j]$.

Step 1 Initially, DeadSegment() checks if $|F| \geq g_k$. If so, it continues; otherwise, it returns FALSE (by the characterization of catastrophic fault patterns given in Chapter 2 [30], the pattern is not catastrophic).

Step 2 Then DeadSegment() scans F to form, for each row, the set of dead segments lying on that row. If, at any time, DeadSegment() encounters a dead segment encompassing the entire row, it returns TRUE. (By Theorem 5.1, the pattern is catastrophic). If all dead segments in F have been built and no dead segment of size g_k is found, DeadSegment() returns FALSE (By Theorem 5.1, the pattern is not catastrophic).

Step 3 The set of dead segments for row(1), $DS[1]$, is composed of the segments formed by the faulty elements in row(1). Given the set $DS[i-1]$ of dead segments for row($i-1$), the set $DS[i]$ of dead segments for row(i) is constructed as follows:

L0: form an initial set $DS[i]$ by merging faulty elements in $row(i)$;

$p=1$;

L1: *if* $DS[i-1]=\emptyset$ *then* $DS[i]$ is done;

$\alpha=|DS[i-1]|$;

for $(1 \leq d \leq k-1)$ $l_d=2$;

for $(1 \leq t \leq \alpha)$ {

$S_k \leftarrow DS[i-1](t)$;

$\Omega \leftarrow$ shadow of S_k at distance g_k ;

for $(k-1 \geq j \geq 1)$ {

$l_j = \max\{l_j, l_{j+1}\}$;

L2: $S_j \leftarrow (DS[i-1] \bullet DS[i])(l_j)$;

$\Omega_j \leftarrow$ shadow of S_j at distance g_j ;

if $\Omega_j \triangleleft \Omega$ *then*

if $l_j = |DS[i-1] \bullet DS[i]|$ *then* goto L3;

else $\{l_j = l_j + 1$; goto L2; $\}$

if $\Omega \triangleleft \Omega_j$ *then* goto L3;

$\Omega \leftarrow \Omega \cap \Omega_j$;

}

MergeSegment($\Omega, DS[i], p$);

L3: continue;

}

and where the MergeSegment() is as follows:

MergeSegment(Ω , DS[i], p)

Input: Ω , DS[i], p;

Output: DS[i], p;

Outline: this procedure first scans DS[i] starting from p to find the location for Ω ; then inserts Ω into DS[i]. The procedure then returns the modified DS[i] and the location of the next segment p.

while ($y_F(\Omega) > y_L(DS[i](p))$) $p = p + 1$;

$y'_F = \min(y_F(\Omega), y_F(DS[i](p)))$;

$q = p$;

while ($y_L(\Omega) > y_F(DS[i](q))$) $q = q + 1$;

$y'_L = \max(y_L(\Omega), y_L(DS[i](q-1)))$;

replace all segments from p to q-1 with the new segment $\{(i, y'_F), (i, y'_L)\}$ in DS[i];

$p = \text{pointer to this new segment}$;

return DS[i] and p.

Note that $p+1$ is the operation of next(p), and $p-1$ is the operation of previous(p).

Example 5.2 The following is an example of Algorithm 5.1. The link redundancy is $G = \{1, 5, 10\}$. The Boolean matrix of the fault pattern F is shown in Figure 5.7(a).

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 5.7(a): Boolean matrix of a fault pattern

As shown in Figure 5.7(b), F is decomposed to F_1 and F_2 since row(3) and row(4) do not have any faulty element.

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

F_1 F_2

Figure 5.7(b): Two separated fault patterns generated

For F_1 , the algorithm $\text{DeadSegment}(F_1)$ first forms, for row(1), a set of dead segments lying on that row by merging continuous faults, as shown in Figure 5.7(c). Since no dead segment is encountered, the algorithm proceeds.

$$\begin{bmatrix} \text{---} & 0 & \text{---} & 0 & 0 & \text{---} & 0 & 0 & 0 \\ \text{+} & 0 & 0 & \text{+---} & 0 & 0 & 0 & 0 & \text{+} \end{bmatrix}$$

Figure 5.7(c): The dead segments generated from the continuous faults in F_1

Then the procedure $\text{DeadSegment}(F_1)$ form $\text{DS}[2]$ by merging continuous faults in $\text{row}(2)$, as shown in Figure 5.7(c). Then it checks each segment of elements directly below each dead segment in $\text{row}(1)$ by checking if the previous 2 segments at distance 1, 5 are dead. In this case, segment $\{(2,3)\}$ will be checked by checking if segments $\{(2,2)\}$ and $\{(1,8)\}$ are all dead. Segment $\{(2,3)\}$ is not dead since segment $\{(2,2)\}$ is not dead. Segment $\{(2,6),(2,7)\}$ will be checked by checking if segments $\{(2,4),(2,5)\}$ and $\{(2,1),(2,2)\}$ are all dead. In this case, the intersection is computed and the result is the first element of segment $\{(2,6),(2,7)\}$. Therefore, there is a new dead segment $\{(2,6)\}$. The original dead segments and the new ones will be merged to enlarge the dead segments, as shown in Figure 5.7(d). Since the end of F_1 is reached and no dead segment of g_k elements is encountered, F_1 is not catastrophic.

Figure 5.7(d): F_1 is not catastrophic

For F_2 , similarly, the procedure $\text{DeadSegment}(F_2)$ constructs all the dead segments in F_2 and there exists a dead segment of size g_k , as shown in Figure 5.7(e). Thus, F_2 is catastrophic. As a result, F is catastrophic.

Figure 5.7(e): A dead segment of g_k elements is encountered

Claim 5.2 Algorithm 5.1 correctly determines whether or not a fault pattern F is catastrophic.

Proof:

F is decomposed into patterns which do not contain any gap. By Theorem 5.2, F is catastrophic if and only if at least one of these patterns is catastrophic. Thus, to prove the correctness, it suffices to prove that the algorithm correctly determines whether any such pattern F' is catastrophic. By Theorem 5.1, F' is catastrophic if and only if there exists a dead segment of size g_k . Thus, it suffices to show that the algorithm correctly constructs all the dead segments; in particular, it suffices to show that $DS[i]$ will be constructed correctly given $DS[i-1]$.

By Definition 3.11, a segment S is dead if all k segments at distance $g_1, g_2, \dots, g_k$ from S are dead. First of all, observe that it suffices to verify the above condition only for the segments which are shadows at distance g_k from the segments in $DS[i-1]$. In fact, any element of $row(i)$, which is a shadow at distance g_k of a live element, is live; in other words, the only possible candidates for dead elements (and, thus, dead segments) are those which are the shadow at distance g_k of a dead segment in $DS[i-1]$.

Let S be a shadow at distance g_k from some segment in $DS[i-1]$. To determine whether an element of S is dead, we must determine whether its shadows at distance $g_1, g_2, \dots, g_{k-1}$ are dead; that is, we must determine whether its shadows at distance $g_1, g_2, \dots, g_{k-1}$ are in $DS[i-1] \bullet DS[i]$. The algorithm first determines which parts of the segments in $DS[i-1] \bullet DS[i]$ are at distance $g_1, g_2, \dots, g_{k-1}$ from S and compute the intersection of all these parts. Any element in S is dead if and only if it belongs to this intersection.

Since this is done for every row, the claim follows. $\square$

Claim 5.3 For any row(i) in the Boolean representation of a fault pattern, let n_i be the number of faults in that row, and $|DS(i)|$ be the number of dead segments in that row, then $|DS(i)| \leq n_i + 1$.

Proof: By contradiction, let $|DS(i)| > n_i + 1$. Then there are at least two dead segments, S_1 and S_2 which do not contain any faults. Without loss of generality, let $S_1 \prec S_2$; thus $y_F(S_2) > 1$. In other words, $w = (i, y_F(S_2) - 1)$ exists and is live. Since S_2 does not contain any faulty element, all elements in S_2 are reachable by p . This contradicts the fact that S_2 is a dead segment. Thus, the statement holds. $\square$

Claim 5.4 Algorithm 5.1 requires $O(kn)$ time, where k is the number of bypass links, and n is the number of faults.

Proof:

The given fault pattern F is decomposed into patterns which do not contain any gap. This is done in time $O(n)$. In the worst case, the procedure **DeadSegment** will be applied to each such pattern.

For any input F' of **DeadSegment**:

- 1) the number of rows in F' is at most $2n'$ where n' is the number of faults in F' . This is because F' does not contain any gap. By Claim 5.3, $|DS(F')|$, the number of dead segments in F' is at most $\sum_{i=1}^{2n'} (n_i + 1) = n' + 2n' = 3n'$. Thus, $|DS(F')|$ is $O(n')$.
- 2) **DeadSegment** construct the overall set DS row by row. It uses $k-1$ pointers (the l_i 's). At each step of the execution, each pointer can move forwards ("advance") or not ("stay"); it cannot ever move backwards. The number of "advances" for each pointer is

at most $|DS(F')|$, so is the number of “stays”. Thus, each pointer requires at most $2|DS(F')| = O(n)$ operations, for a total of $O(kn)$ time for $k-1$ pointers.

- 3) As for the cost of the merging operation, the initial merge (executed in step L0 of the algorithm) requires $O(n_i')$ operations. The total cost of merging new dead segments with $DS[i]$ by calling **MergeSegment** is $O(n_i')$ since the entire $DS[i]$ is only scanned once. The total cost for F' is $O(n')$.

Therefore, the total cost for F' is $O(kn')$.

Thus, the total cost for each pattern i'' into which F has been decomposed is $O(kn')$. Since these patterns are disjoint, the total cost of Algorithm 5.1 is $O(kn)$, where k is the number of bypass links and n is the number of faults. $\square$

Theorem 5.3 Algorithm 5.1 correctly determines whether or not a fault pattern is catastrophic in $O(kn)$ time, where k is the number of bypass links, n is the number of faults.

Proof: The theorem follows Claims 5.2 and 5.4. $\square$

5.4 Chapter Summary

In this chapter, the problem of determining a fault pattern to be catastrophic for unidirectional linear array with k -link redundancy has been studied. First we establish a necessary and sufficient condition for a fault pattern to be catastrophic. Based on this necessary and sufficient condition, we derive an efficient testing algorithm whose complexity is $O(kn)$, where k is the number of bypass links, and n is the number of faults.

Chapter 6

Summary and Conclusions

6.1 Summary of Main Contributions

The results of this thesis improve the understanding of fault tolerance and fault intolerance in redundancy-based systolic arrays. A different approach (i.e., geometric approach) from the ones adopted in previous work is used in this thesis to characterize and test fault patterns in linear arrays with redundant unidirectional links.

The main contributions of this thesis are:

- (1) a geometric characterization of fault patterns and,
- (2) two testing schemes of fault patterns in linear unidirectional arrays.

Fault patterns are geometrically characterized by means of geometric objects: such as rectangles, staircases and segments are introduced to represent dead objects consisting of either faulty PE(s) or non-faulty PE(s) which are unusable in any computation. The geometric characterization reveals several important properties with respect to these geometric dead objects. One of the important properties emphasizes the point that the dead objects actually block all data flows across the dead objects. Algorithms are provided to identify all geometric dead objects so that any future reconfiguration scheme can make use of this valuable knowledge.

The difference between this characterization and the existing characterizations is that this characterization deals with more general case where the number of faults is arbitrary, using a completely different approach.

Given a fault pattern, it is crucial for any reconfiguration scheme to know whether a fault pattern is catastrophic. If the fault pattern is catastrophic, there is no sense for the reconfiguration scheme to proceed. Therefore, the testing of catastrophe of a fault pattern is a very useful thing for any reconfiguration scheme. Based on the geometric characterization of fault patterns, two testing schemes are developed to test whether a given fault pattern is catastrophic for a given link redundancy.

The first testing algorithm, given in Chapter 4, deals with fault patterns in unidirectional linear arrays with 2-link redundancy. The time complexity of the algorithm is $O(n^3)$ where n is the number of faults.

The second testing algorithm, presented in Chapter 6, copes with fault patterns in unidirectional linear arrays with arbitrary k -link redundancy. This is an efficient testing algorithm since its complexity is $O(kn)$, where k is the number of redundant links and n is the number of faults. In case of 2-link redundancy where $k=2$, the complexity is only $O(n)$ which is a significant improvement of $O(n^3)$, the complexity of the first testing algorithm. Furthermore, the complexity of the second algorithm, $O(kn)$, improves $O(kn \log k)$, the complexity of the testing scheme by [97].

6.2 Applications

The contribution of the thesis can be judged from its theoretical importance, practical relevance, and its applicability to the existing architectures.

From a theoretical viewpoint, the problem of deciding whether if a fault pattern is catastrophic for a given link redundancy is equivalent to the problem of deciding whether a set of nodes is cut set for a certain type of connected graph. The testing schemes developed in this thesis can be directly applied to the equivalent problem in graph theory.

From a practical viewpoint, the results can answer questions about fault tolerance of a design, such as i) will the system withstand a distribution of faults? ii) if the system is able to withstand the distribution of faults, what elements can be employed for the future reconfiguration, and what elements cannot?

The results presented in this thesis apply to a large variety of commercially available array processors such as Geometric Arithmetic Parallel Processor (GAPP) [20] of NCR, Distributed Array Processor (DAP) of ICL, England, NASA's Massively Parallel Processor (MPP) [15], and Connection Machine of Thinking Machines Corporation in identifying the answers to questions regarding fault intolerance. The results also apply to a large number of WSI-based and VLSI-based processor arrays which include the Systolic Arrays [66], Reconfigurable Array of Processors ELSA (European large SIMD array), and a variety of special-purpose VLSI and experimental WSI devices for applications such as signal processing, and numerical computations. Furthermore, the results are equally applicable to the design of fault-tolerant memory chips[36].

6.3 Areas of Future Work

This thesis only deals with fault patterns in linear unidirectional systolic arrays. Therefore, the problem of using similar geometric characterization to study fault patterns in linear bidirectional systolic arrays is left open. Furthermore, geometric approach can also be applied to fault patterns in two dimensional systolic arrays and chordal rings[4].

Only the PE failures are considered in this thesis. The results can still be directly applied in the presence of link failures (e.g., by inscribing the failure of a link to one of its incident PEs, by considering the dual graph of the network topology, etc.). A future direction for research is to extend and generalize the model so that link failures can be included explicitly.

The work in this thesis can be easily extended to study fault patterns in bidirectional linear arrays with redundant links and develop some testing algorithm.

This thesis only solved the problem of recognizing catastrophic fault patterns. The problem of counting the number of catastrophic fault patterns given the number of faults and the link configuration is still open.

Reference

- [1] V. Acciaro, A. Nayak, and N. Santoro, "On the Fault-Tolerance of Reconfigurable VLSI Systolic Arrays," *Congressus Numerantium*, Vol. 68, May 1989, pp.197-210.
- [2] V. Acciaro, A. Nayak, and N. Santoro, "Catastrophic Faults for Reconfigurable Systolic Array." Tech. Report, Centre for Parallel and Distributed Computing, Carleton University, Ottawa, Canada, 1989.
- [3] A. Aggarwal and J. Park, "Parallel Searching in Multidimensional Monotone Arrays," *Proc. 29th Annual IEEE Symposium on Foundations of Computer Science*, 1988, pp. 497-512.
- [4] A. Apostolico, M. J. Atallah, L. Larmore, H. S. McFaddin, "Efficient Parallel Algorithms for String Editing and Related Problems," *SIAM J. Comput.*, Vol. 19, 1990, pp. 968-998
- [5] M. J. Atallah, "A Faster Parallel Algorithm for a Matrix Searching Problem," *Algorithmica*, 1993, pp. 156-167.
- [6] M. J. Atallah and S. R. Kosaraju, "An Efficient Parallel Algorithm for the Row Minima of a Totally Monotone Matrix," *J. of Algorithms*, 1992, pp. 394-413.
- [7] K.P. Belkhale and P. Banerjee, "Reconfiguration Strategies in VLSI Processor Arrays," in *Proc. Int'l Conf. on Computer Design*, 1988, pp. 418-421.
- [8] M. Chean and J. A. B. Fortes, "A Taxonomy of Reconfiguration Techniques for Fault-Tolerant Processor Arrays," in *IEEE Computer*, Vol. 23, No. 1, Jan. 1990, pp.55-69.
- [9] F. Distanto, F. Lombardi, and D. Sciuto, "Array Partitioning: A Methodology for Reconfigurability and Reconfiguration Problems," in *Proc. Int'l Conf. on Computer Design*, 1988, pp.564-567.
- [10] J. A. B. Fortes and C. S. Raghavendra, "Dynamically Reconfigurable Fault-Tolerant Array Processors," in *Proc. 14th Int'l Conf. on Fault-Tolerant Computers*, June 1984, pp.386-392.

- [11] J. W. Greene and A. E. Gamal, "Configuration of VLSI Arrays in the Presence of Defects," *Journal of the ACM*, Vol. 31, No. 4, 1984, pp.694-717.
- [12] N. Hasan and C. L. Liu, "Minimum Fault Coverage in Reconfigurable Arrays," in *Proc. 18th Int'l Conf. on Fault-Tolerant Computers*, 1988, pp.348-353.
- [13] J. P. Hayes, "A Graph Model for Fault-Tolerant Computing Systems," *IEEE Trans. on Computers*, Vol. C-25, No. 9, Sept. 1976, pp. 875-884.
- [14] S. H. Hosseini, "On Fault-Tolerant Structure, Distributed Fault-Diagnosis, Reconfiguration, and Recovery of the Array Processors," *IEEE Trans. on Computers*, Vol. C-38, No. 7, July 1989, pp. 932-942.
- [15] J. H. Hwang and C. S. Raghavendra, "VLSI Implementation of Fault-Tolerant Systolic Arrays," in *Proc. Int'l Conf. on Computer Design*, 1986, pp. 110-113.
- [16] I. Koren, "A Reconfigurable and Fault-Tolerant VLSI Multiprocessor Array," in *Proc. Symp. Computer Architecture*, 1981, pp.425-442.
- [17] I. Koren and D.K. Pradhan, "Introducing Redundancy into VLSI Designs for Yield and Performance Enhancement," in *Proc. 15th Int'l Conf. on fault-Tolerant Computers*, 1985, pp. 330-335.
- [18] I. Koren and D. K. Pradhan, "Yield and Performance Enhancement Through Redundancy in VLSI and WSI Multiprocessor Systems," *Proc. of IEEE*, Vol. 74, NO. 5, May 1986, pp. 699-711.
- [19] I. Koren and I. Pomeranz, "Distributed Structuring of Processor Arrays in the Presence of Fault Processors," in *Proc. Int'l Conf. on Systolic Arrays*, 1986, pp. 239-248.
- [20] R. Kumar and F.G. Gray, "Fault Tolerant 1-Dimensional Cellular Structure," in *1984 Conf. on Distributed Computing*, May 1984, pp. 472-483.
- [21] H. T. Kung, "Why Systolic Architecture?," *IEEE Computer*, Vol. 15, No. 1, Jan. 1982, pp. 37-46.

- [22] H. T. Kung and M. Lam, "Fault-Tolerant VLSI Systolic Arrays and Two-level Pipelining," *Journal of Parallel and Distributed Processing*, Aug. 1984, pp. 32-63.
- [23] S. Y. Kung, C. W. Chang, and C. W. Jen, "Fault-Tolerance Design in Real-Time VLSI Array Processors," in *Proc. Conf. on Computers and Communications*, Phoenix, 1987, pp. 110-115.
- [24] S. Y. Kung, D. D'Souza, and J. T. Juhl, "On Fault-Tolerance in Array Processors," in *Proc. Int'l Conf. on Computer Design*, 1985, pp. 764-768.
- [25] S. Kuo and W. K. Fuchs, "Efficient Spare Allocation for Reconfigurable Arrays," *IEEE Design and Test*, Feb. 1987, pp. 24-31.
- [26] C. W. H. Lam, H. F. Li, and R. Jayakumar, "A Study of Two Approaches for Reconfiguring Fault-Tolerant Systolic Arrays," *IEEE Trans. on Computers*, Vol. C-36, No. 6, June 1989, pp. 833-844.
- [27] T. Leighton and C. E. Leiserson, "Wafer-Scale Integration of Systolic Arrays," *IEEE Trans. on Computers*, Vol. C-34, No. 5, May 1985, pp. 448-461.
- [28] H. F. Li, R. Jayakumar, and C. Lam, "Restructuring for Fault-Tolerant Systolic Arrays," *IEEE Trans on Computers*, Vol. C-38, No. 2, Feb. 1989, pp. 307-311.
- [29] F. Lombardi, R. Negrini, M. Sami, and R. Stefanelli, "Reconfiguration of VLSI Arrays: A Covering Approach," in *Proc. 17th Int'l Conf. on Fault-Tolerant Computers*, 1987, pp. 251-256.
- [30] A. Majumdar, C. S. Raghavendra and M. A. Breuer, "Fault Tolerance in Linear Systolic Arrays Using Time Redundancy," *IEEE Trans. on Computers*, Vol. C-39, No. 2, Feb. 1990, pp. 269-276.
- [31] T. E. Mangir and C. S. Raghavendra, "Issues in the Implementation of Fault Tolerant VLSI and WSI Systems," in *Proc. Int'l Conf. on Computer Design*, 1984, pp. 95-100.
- [32] W. Marwood and A. P. Clarke, "Fault Tolerant Systolic Architectures," in *3rd National Workshop on Fault Tolerant Computing*, Melbourne, Australia, July 1985.

- [33] W. R. Moore, "A Review of Fault-Tolerant Techniques for the Enhancement of Integrated Circuit Yield," *Proc. of the IEEE*, Vol. 74, No.5, May 1986, pp.684-698.
- [34] A. Nayak, "On Reconfigurability of some Regular Architectures," Ph.D Thesis, Dept. System & Computer Engineering, Carleton University, Ottawa, Canada.
- [35] A. Nayak, L. Pagli and N. Santoro, "Efficient Construction of Catastrophic Patterns for VLSI Reconfigurable Arrays with Bidirectional Links," in *Proc. 4th Int'l Conf. on Computing and Information*, 1992, pp. 79-83.
- [36] A. Nayak, L. Pagli and N. Santoro, "Recognition of Catastrophic Faults in Reconfigurable Arrays with Arbitrary Link Redundancy," Tech. Report, Centre for Parallel and Distributed Computing, Carleton University, Ottawa, Canada, 1992.
- [37] A. Nayak, L. Pagli, N. Santoro, "Combinatorial and Graph Problem Arising in the Analysis of Catastrophic Fault Patterns," in *Congressus Numerantium*; November, 1992, pp. 7-22.
- [38] A. Nayak, N. Santoro and R. Tan, "Fault-Intolerance of Reconfigurable Systolic Arrays," in *20th Int'l Conf. on Fault-Tolerant Computers*, 1990, pp. 202-209.
- [39] A. Nayak and N. Santoro, "Fault-Tolerance Analysis of some Regular Networks," in *Int'l Conf. on Parallel Processing Technology(PARCOM-90)*, Pune, India, Dec. 1990, pp. 67-76.
- [40] A. Nayak and N. Santoro, "Bounds on Performance of VLSI Processor Arrays," in *5th Int'l Parallel Processing Symposium*, Anaheim, California, May 1991.
- [41] A. Nayak, N. Santoro and R. Tan, "On Reconfigurability of Systolic Arrays," Tech. Report, Centre for Parallel and Distributed Computing, Carleton University, Ottawa, Canada, 1991.
- [42] R. Negrini, M. Sami, and R. Stefanelli, "Fault-Tolerance Approaches for VLSI/WSI Arrays," in *Proc. Conf. on Computers and Communications*, Phoenix, 1985, pp. 460-468.

- [43] R. Negrini, M. Sami, and R. Stefanelli, "Fault-Tolerance Techniques for Array Structures used in Supercomputing," *IEEE Computer*, Vol. 19, No. 2, Feb. 1986, pp. 78-87.
- [44] S. Pateras and J. Rajske, "A Self-Reconfiguration Scheme for Fault-Tolerant VLSI Processor Arrays," in *Proc. Int'l Conf. on Computer Design*, 1988, pp. 560-563.
- [45] V. Piuri and R. Stefanelli, "use of Redundant Binary Representation for Fault-Tolerant Array Processors," in *Proc. Int'l Conf. on Computer Design*, 1988, pp. 496-501.
- [46] S. P. Popli and M. A. Bayoumi, "Fault Diagnosis and Reconfiguration for Reliable VLSI Arrays," in *Proc. Conf. on Computers and Communications*, Phoenix, 1988, pp. 69-73.
- [47] D. K. Pradhan, Ed., *Fault-Tolerant Computing, Theory and Techniques*, Vol. 1 and 2, Englewood Cliff, NJ: Prentice-Hall, 1986.
- [48] R.D. Prisco, A. Monti, "On reconfigurability of VLSI linear arrays", in *Proceedings of the 3rd Workshop on Algorithms and Data Structures*, Montreal, August, 1993. In *Lecture Notes in Computer Science*, Springer-Verleg, to appear.
- [49] C. S. Raghavendra and T. E. Mangir, "On the VLSI Implementation of Fault Tolerant Architectures," in *Proc. Int'l Conf. on Computer Design*, 1983, pp. 744-747.
- [50] C. S. Raghavendra, "Fault Tolerance in Regular Network Architectures," *IEEE Micro*, Vol. 4, No. 6, Dec. 1984, pp. 44-53.
- [51] C. V. Ramamoorthy and Y. W. Eva Ma, "Optimal Reconfiguration Strategies for Reconfigurable Computer Systems with no Repair," *IEEE Trans. on Computers*, Vol. C-35, No. 3, Mar. 1986, pp. 278-280.
- [52] D. A. Rennels, "Fault-Tolerant Computing- Concepts and Examples," *IEEE Trans. on Computers*, Vol. C-33, No. 12, Dec. 1984, pp. 1116-1129.
- [53] A. L. Rosenberg, "The Diogenes Approach to Testable Fault-Tolerant Arrays of Processors," *IEEE Trans. on Computers*, Vol. C-32, No. 10, Oct. 1983, pp. 902-910.

- [54] A. L. Rosenberg, "On Designing Fault-Tolerant VLSI Processor Arrays," *Advances in Computer Research*, 1984, pp. 181-204.
- [55] A. L. Rosenberg, "Graph Theoretical Approach to Fault-Tolerant Processor Arrays," in *Int'l Workshop on Wafer Scale Integration*, Southampton, 1985, pp. 1-16.
- [56] V. P. Roychowdhury, J. Bruck and T. Kailath, "Efficient Algorithms for Reconfiguration in VLSI/WSI Arrays," *IEEE Trans. on Computers*, Vol. C-39, No. 4, Apr. 1990, 480-489.
- [57] M. G. Sami and R. Stefanelli, "Reconfigurable Architectures for VLSI Processing Arrays," in *Proc. NCC 83*, Los Angeles, May 1983, pp. 567-577.
- [58] M. G. Sami and R. Stefanelli, "Fault-Tolerance of VLSI Processing Arrays: The Time Redundancy Approach," in *Proc. Real-Time Systems Symp.*, Dec. 1984, pp. 200-207.
- [59] M. G. Sami and R. Stefanelli, "Fault-Stealing: An Approach to Fault-Tolerance of VLSI Array Structure," in *Proc. Int'l Conf. on Circuits and Systems*, 1985, pp. 205-210.
- [60] M. G. Sami and R. Stefanelli, "Reconfigurable Architectures for VLSI Processing Arrays," *Proc. of the IEEE*, Vol. 74, No. 5, May 1986, pp. 712-722.
- [61] D. P. Siewiorek, "Fault Tolerance in Commercial Computers," in *IEEE Computers*, Vol. 23, No. 7, July 1990, pp. 26-37.
- [62] C. H. Stapper, F. M. Armstrong, and K. Saji, "Integrated Circuit Yield Statistics," *Proc. of the IEEE*, Vol. 71, No. 4, April 1983, pp. 453-470.
- [63] R. Tamassia and J. S. Vitter, "Parallel Transitive Closure and Point Location in Planar Structures," *SIAM J. Comput.* 20(4), 1991, pp. 708-725.
- [64] J. Tyszer, "A Multiple Fault-Tolerant Processor Network Architecture for Pipelining Computing," *IEEE Trans. on Computers*, Vol. C-37, No. 11, Nov. 1988, pp. 1414-1418.

- [65] M. Wang, M. Cutler, and S. Y. H. Su, "Reconfiguration of VLSI/WSI Array Processors with Two-Level Redundancy," *IEEE Trans. on Computers*, Vol. C-38, No. 4, April 1989, pp. 547-554.
- [66] S. Yalamanchili and J. K. Aggarwal, "Reconfiguration Strategies for Parallel Architectures," *IEEE Computer*, Vol. 18, No. 12, Dec. 1985, pp. 44-61.
- [67] H. Y. Youn and A. D. Singh, "A Highly Efficient Design for Reconfiguring the Processor Array in VLSI," in *Proc. Int'l Conf. on Parallel Processing*, 1988, pp. 375-382.

END

1 9-0 7-9 4

FIN