INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
Fault diagnosis and yield enhancement in defect tolerant VLSI/WSI parallel architectures

Wang, Kuochen, Ph.D.
The University of Arizona, 1991
FAULT DIAGNOSIS AND YIELD ENHANCEMENT IN
DEFECT TOLERANT VLSI/WSI PARALLEL ARCHITECTURES

by

Kuochen Wang

A Dissertation Submitted to the Faculty of the
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
In Partial Fulfillment of the Requirements
For the Degree of
DOCTOR OF PHILOSOPHY
WITH MAJOR IN ELECTRICAL ENGINEERING
In the Graduate College
THE UNIVERSITY OF ARIZONA
1991
As members of the Final Examination Committee, we certify that we have read the dissertation prepared by Kuochen Wang entitled FAULT DIAGNOSIS AND YIELD ENHANCEMENT IN DEFECT TOLERANT VLSI/WSI PARALLEL ARCHITECTURES and recommend that it be accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy.

Sy-Yen Kuo  
Fredrick J. Hill  
Jerzy W. Rozenblit

Final approval and acceptance of this dissertation is contingent upon the candidate's submission of the final copy of the dissertation to the Graduate College.

I hereby certify that I have read this dissertation prepared under my direction and recommend that it be accepted as fulfilling the dissertation requirement.

Dissertation Director Sy-Yen Kuo
STATEMENT BY AUTHOR

This dissertation has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under the rules of the library.

Brief quotations from this dissertation are allowable without special permission, provided that accurate acknowledgement of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgement the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: [Signature]

[Author's Name]
To My Wife, My Mother, and to the Memory of My Father
ACKNOWLEDGMENTS

The author wishes to express his deepest gratitude to his dissertation advisor, Dr. Sy-Yen Kuo, for his support, encouragement, and insightful guidance throughout this dissertation work.

The author would also like to thank Dr. Fredrick J. Hill, Dr. Jerzy W. Rozenblit, Dr. Olivia R. Liu Sheng, and Dr. Hsinchun Chen for their advice and time, and all his colleagues in the Fault Tolerant Computing Group for their friendship and intellectual stimulation. Special thanks go to Mrs. Becky Gentner for her helpful suggestions. The support by the National Science Foundation under Grant MIP-89-08648 is also gratefully acknowledged.

The author wishes to thank his mother and family whose love and support made it possible to complete this dissertation. Last but not least, he would like to express his appreciation to his wife, Maisying, for her continual encouragement, understanding, faith, and patience throughout his Ph.D. studies.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>LIST OF FIGURES</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF TABLES</td>
<td>11</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>12</td>
</tr>
<tr>
<td><strong>1. INTRODUCTION</strong></td>
<td>14</td>
</tr>
<tr>
<td>1.1. Objective</td>
<td>14</td>
</tr>
<tr>
<td>1.2. Motivation</td>
<td>15</td>
</tr>
<tr>
<td>1.3. Overview of the CAD Framework</td>
<td>16</td>
</tr>
<tr>
<td>1.4. Overview of the Array Model and Its VHDL Description and Simulation</td>
<td>17</td>
</tr>
<tr>
<td>1.5. Overview of the Fault Diagnosis Techniques</td>
<td>17</td>
</tr>
<tr>
<td>1.6. Overview of the Simulation and Evaluation Processes</td>
<td>18</td>
</tr>
<tr>
<td>1.7. Applications</td>
<td>18</td>
</tr>
<tr>
<td><strong>2. CAD FRAMEWORK</strong></td>
<td>19</td>
</tr>
<tr>
<td>2.1. Introduction</td>
<td>19</td>
</tr>
<tr>
<td>2.2. VAR Organization</td>
<td>21</td>
</tr>
<tr>
<td>2.3. Summary</td>
<td>26</td>
</tr>
<tr>
<td><strong>3. ARRAY MODEL AND VHDL DESCRIPTION</strong></td>
<td>27</td>
</tr>
<tr>
<td>3.1. Introduction</td>
<td>27</td>
</tr>
<tr>
<td>3.2. Array Model and Its Implication</td>
<td>27</td>
</tr>
<tr>
<td>3.3. Array Design and Its VHDL Description</td>
<td>30</td>
</tr>
<tr>
<td>3.3.1. System Design</td>
<td>30</td>
</tr>
<tr>
<td>3.3.2. PE Design</td>
<td>35</td>
</tr>
<tr>
<td>3.3.3. Switch Design</td>
<td>37</td>
</tr>
<tr>
<td>3.4. Comparison with an Existing Modeling Method</td>
<td>39</td>
</tr>
<tr>
<td>3.5. Summary</td>
<td>41</td>
</tr>
<tr>
<td><strong>4. FAULT DIAGNOSIS</strong></td>
<td>42</td>
</tr>
<tr>
<td>4.1. Parallel Partition Approach</td>
<td>42</td>
</tr>
<tr>
<td>4.1.1. Introduction</td>
<td>42</td>
</tr>
</tbody>
</table>
7.1. Summary ........................................................................................................... 134
7.2. Suggested Future Research .............................................................................. 136

REFERENCES ........................................................................................................... 138
LIST OF FIGURES

2.1. VAR system configuration. ................................................................. 22
3.1. A PE-switch lattice model. ................................................................. 28
3.2. Classification used in the characteristic matrix. .......................... 30
3.3. Hierarchical VHDL description of the reconfigurable array. ....... 31
3.4. VHDL test bench environment. ....................................................... 32
3.5. VHDL entity and component declarations of the test bench environment. .......... 33
3.6. Two PE structures. ........................................................................... 36
3.7. VHDL entity declaration of $PE_b$. .................................................. 37
3.8. Switch structure. ............................................................................. 38
3.9. Switch states. .................................................................................. 39
3.10. VHDL entity declaration of the switch. ......................................... 40
4.1. A $5 \times 6$ logic array and its diagonal groups. ......................... 47
4.2. A general PE model. ....................................................................... 48
4.3. Parallel partition algorithm. ............................................................. 53
4.4. Three typical PEs in systolic arrays. .............................................. 54
4.5. Testable designs of the three PEs in Figure 4.4. ......................... 55
4.6. Selector functions. ......................................................................... 56
4.7. Testing and diagnosis process. ....................................................... 58
4.8. Multiple PE-faults diagnosis algorithm. .......................................... 65
4.9. A host array and its diagnosis matrices. ........................................ 66
4.10. Example parallel architectures. ..................................................... 69
4.11. A two-dimensional array. ............................................................... 74
4.12. A PE configuration model. ............................................................. 74
4.13. Selector connection models. ............................................................ 75
4.14. WSI production process. ................................................................. 78
4.15. Fault diagnosis algorithm. ............................................................... 80
4.16. Error matrix processing. ................................................................. 81
5.1. An example logic array. ................................................................. 90
5.2. A 4-ary decision tree. ................................................................. 91
5.3. A binary decision tree. ................................................................. 95
5.4. A block-walking representation. .................................................... 101
5.5. Integrated diagnosis and reconfiguration procedure. .................. 103
5.6. Classification of diagnosis strategies. ................................................................. 104
5.7. Comparison of ADCs between OGD and OSD (Y_{av} = 0.50). ....................... 106
5.8. ADC versus Y_{av}. ............................................................................................ 108
5.9. ADC_{cs} versus Y_{av}. .................................................................................... 110
6.1. Reconfiguration process in VAR. ...................................................................... 113
6.2. Pictorial view of the reconfiguration pattern mapping. ..................................... 114
6.3. Symbolic representation of the reconfiguration process. .................................. 116
6.4. Graphical representation of the reconfiguration process. ............................... 117
6.5. A matrix multiplication. .................................................................................. 118
6.6. Array evaluation. ............................................................................................. 127
6.7. Reliability evaluation. ...................................................................................... 132
LIST OF TABLES

4.1. Speed-up summary for various parallel architectures. .............................................. 70
5.1. Yield and diagnosis cost for the example logic array. .............................................. 90
5.2. $ADC$ comparison with $Y_{av} = 0.30, 0.65$ .......................................................... 107
5.3. $ADC$ related statistics with $Y_{av} = 0.30, 0.50, 0.65$. .......................................... 109
6.1. Simulation results of the $PE_b$-based array. ........................................................... 120
6.2. Run time for the $PE_a$ and $PE_b$-based arrays. ..................................................... 121
6.3. Redundancy evaluation under the random fault distribution. ................................. 129
6.4. Redundancy evaluation under the clustered fault distribution. .............................. 130
6.5. Reconfiguration algorithm evaluation. ................................................................. 131
ABSTRACT

This dissertation presents an integrated high-level computer-aided design (CAD) environment, the VAR (VHDL-based Array Reconfiguration) system, for the tasks of design, diagnosis, reconfiguration, simulation, and evaluation in a defect tolerant VLSI/WSI (Wafer Scale Integration) parallel architecture modeled by VHDL. Four issues in the VAR system are studied: (1) the development of a CAD framework for reconfigurable architectures, (2) the development of an array model, and its VHDL description and simulation, (3) the development of efficient fault diagnosis techniques, and (4) the development of a systematic method for evaluating architectures and yield. The first issue describes the modules in the CAD framework and their functionalities. The second issue addresses the hierarchical VHDL description and simulation of the array model, and the detailed designs of its components. The third issue proposes two fault diagnosis algorithms based on the parallel partition approach and the self-comparison approach respectively, and an optimal group diagnosis procedure. These fault diagnosis techniques all have the contribution of reducing testing time significantly under different application scenarios. The fourth issue depicts a complete set of figures of merits for quantitative architecture and yield evaluation.

Although an easily diagnosable and reconfigurable two-dimensional defect tolerant array is used as an example to illustrate the methodology of VAR, the VAR environment can be equally applied to other parallel architectures. VAR allows the designers to study and evaluate fault diagnosis and reconfiguration algorithms by inserting faults, which are generated according to
actual manufacturing yield data, into the array and then locating the faulty elements as well as simulating the reconfiguration process. Thus, VAR can assist the designers in evaluating different combinations of fault patterns, fault diagnosis and reconfiguration techniques, and reconfigurable architectures through the figures of merit with aim at architectural improvements. Extensive simulation and evaluation have been performed to demonstrate and support the effectiveness of VAR. The results from this research can drive the applications of large area VLSI or WSI closer to reality and result in producing low cost and high yield parallel architectures.
CHAPTER 1.

INTRODUCTION

1.1. Objective

The objective of this research is development of an integrated high-level computer-aided design (CAD) environment, the VHDL-based Array Reconfiguration (VAR) system, for the design and evaluation of reconfigurable VLSI or WSI (wafer scale integration) parallel architectures. With the rapid progress of VLSI technology, the chip sizes are to increase substantially, perhaps up to the level of WSI. However, low manufacturing yield has become a problem of increasing significance, especially in WSI where it is unlikely to get a defect-free wafer.

Redundancy techniques are utilized to overcome this difficulty. Substantial yield and reliability improvements in VLSI/WSI parallel architectures can be achieved by an appropriate combination of architectural redundancy, fault diagnosis, and reconfiguration techniques. The emphasis is on the design and analysis of application specific reconfigurable architectures, and fault diagnosis and reconfiguration algorithms for large area VLSI or WSI parallel architectures. Successful results from this research can make the implementation of large area VLSI/WSI architectures feasible.
1.2. Motivation

Increase in the transistor count of an integrated circuit has been achieved to date primarily by reducing the dimensions of a transistor, and by using more densely packed design styles. However, the technology is being pushed to its physical limits, and further increase in the integration size may have to come from the increase in die area which can be up to a whole wafer (wafer scale integration). The yield is a crucial parameter in the semiconductor manufacturing process and decreases rapidly as the circuit area increases. Design for yield enhancement techniques are therefore necessary to increase manufacturing feasibility. One important method of increasing yield is to incorporate hardware redundancy and switching mechanisms into a chip or wafer.

Design and analysis of reconfigurable VLSI parallel architectures is an increasingly complicated task, especially in evaluating the impact of reconfiguration on performance, overhead, and yield. It is therefore necessary to develop such an integrated high-level CAD environment [1] to assist in simulating and evaluating various combinations of fault patterns, fault diagnosis techniques, reconfiguration techniques, and reconfigurable architectures. This environment will enable system designers to reduce the design turnaround time, to pinpoint possible design problems in the early design phase, and to optimize their designs. Spare allocation and reconfiguration [2, 3] are not possible without the detection and location of faults. Fault diagnosis is a prerequisite to successful reconfiguration, but is extremely difficult in VLSI/WSI due to low controllability and observability. Several on-line fault diagnosis methods applicable to VLSI systems have been proposed [4, 5]. These methods are based on either time or hardware redundancy to detect errors and identify faulty elements. However, there is significantly little published research on off-line diagnosis in reconfigurable VLSI and WSI designs. We propose
two efficient diagnosis algorithms based on the parallel partition approach and the self­
comparison approach respectively. An optimal group diagnosis procedure is also proposed,
which is again based on the parallel partition approach, and aims at further testing speed-up by
integrating the fault diagnosis process and the reconfiguration process.

The goal of this research is to demonstrate that substantial yield improvement can be
achieved by a proper integration of architectural redundancy, fault diagnosis, and
reconfiguration techniques. Specifically, the four issues addressed in this dissertation are as fol­
lows:

• Development of a CAD framework for reconfigurable architectures,

• Development of an array model and its VHDL description and simulation,

• Development of efficient fault diagnosis techniques, and

• Development of a systematic method for evaluating architectures and yield.

In the following, these four issues are overviewed and their arrangements in the dissertation are
outlined.

1.3. Overview of the CAD Framework

The CAD framework, the VAR system, for defect tolerant architectures is presented in
Chapter 2. The system consists of the following major modules: the fault diagnosis module, the
reconfiguration module, the evaluation module, the optimization module, and the synthesis
module. The function of each module and the interaction between modules are described.
1.4. Overview of the Array Model and Its VHDL Description and Simulation

The array model and its VHDL description is presented in Chapter 3. The details of the array design, PE (processing element) design, switch design, and their VHDL descriptions are depicted. In order to simulate different component implementations, two types of PE structures are designed. The VHDL simulation of reconfigurable arrays is discussed in Chapter 6. Two arrays with these two types of PE structures are simulated to evaluate their pros and cons.

1.5. Overview of the Fault Diagnosis Techniques

The fault diagnosis techniques are presented in Chapters 4 and 5. In Chapter 4, we propose two efficient fault diagnosis algorithms based on the parallel partition approach and the self-comparison approaches respectively. These two approaches are suitable for the instances that we need to identify all the faulty elements first before a repair action can be taken. That is, we need to know the overall fault distribution in an architecture in order to obtain a better reconfiguration result in terms of interconnection length and spare utilization. In Chapter 5, we present an optimal group diagnosis procedure which is also based on the parallel partition approach with aim at further testing speed-up by choosing an optimal testing sequence. In order to take advantage of the optimal group diagnosis procedure, the integrated diagnosis and reconfiguration is discussed.
1.6. Overview of the Simulation and Evaluation Processes

In Chapter 6, the reconfiguration process is simulated. The reconfiguration process interfaces the reconfiguration program, the switching mechanism transformation program, and the VHDL simulation program for target array generation and array functionality verification. The VHDL simulation results are analyzed. Then, we present a complete set of figures of merit to evaluate reconfigurable architectures. Extensive simulation is preformed and experimental results are evaluated according to different combinations of the fault pattern, redundancy scheme, switching mechanism, and reconfiguration algorithm.

1.7. Applications

The increasing demands of high throughput and reliability in modern computer applications necessitate a revolutionary supercomputing technology. The availability of low-cost, high density large area VLSI and WSI devices and the emerging CAD tools will make a major breakthrough in the design and application of massively parallel architectures. A single wafer may contain hundreds or thousands of PEs interconnected in a near-neighbor structure by employing some defect tolerant schemes. These architectures have applications in a wide variety of areas including signal and image processing, computer-aided design, artificial intelligence, and other engineering disciplines. The research in this dissertation will help speed up and strengthen the development in this direction.
CHAPTER 2.

CAD FRAMEWORK

2.1. Introduction

In this chapter, we present an integrated high-level CAD environment, VHDL-based Array Reconfiguration (VAR) system for the tasks of design, diagnosis, reconfiguration, simulation, and evaluation on an architecture described in a hardware description language, VHDL. VHDL, the IEEE 1076 Standard is the basis of the VAR system. It is used to model reconfigurable parallel architectures in a modular way, and its support environment is used to simulate and evaluate different design alternatives. We will use array architectures to illustrate this system. One of the advantages of developing such a VHDL-based tool is making other VHDL-based tools available to VAR and vice versa. VHDL has been adopted as a standard hardware description language by the U. S. Department of Defense and is gaining popularity in industries and academia. We can use manufacturing yield data to generate different fault patterns to see if a simulated array has the capability of withstanding the possible defects through fault diagnosis and reconfiguration. The above results, along with the quantitative evaluation of the target array, can help the designers determine appropriate redundancy deployment and allocation. The procedure of modeling, simulation and evaluation processes is illustrated by an example two-dimensional reconfigurable systolic array. The transformation of a faulty array into a target array in a VHDL support environment is used to simulate the actual reconfiguration process. The correctness of the reconfiguration process and the functionality of the target array are
verified by performing matrix multiplications. Experimental results are included to demonstrate
the evaluation process. The issues evaluated include redundancy overhead, efficiency of
reconfiguration algorithms, fault distribution effect, array quality, yield, and reliability.

Although extensive research in the areas of fault diagnosis and reconfiguration [6, 7, 8, 9]
has been performed, integrated architecture-level computer-aided design (CAD) tools are lack­ing in assisting the design and evaluation of defect-tolerant architectures. Many physical level
VLSI CAD tools, such as tools for layout design automation, are available today, but few are
for architecture-level system design [10], especially for reconfigurable architectures. As we
mentioned before, design and analysis of reconfigurable VLSI parallel architectures is an
increasingly complex task. Developing such an integrated high-level CAD environment to
assist in simulating and evaluating various combinations of fault patterns, fault diagnosis and
reconfiguration techniques, and reconfigurable architectures is therefore very necessary. This
environment will enable system designers to experiment with different design options through
simulation and evaluation for architectural improvements. It will drive the applications of large
area VLSI or WSI (wafer scale integration) closer to reality and help produce low cost and high
yield parallel architectures. The paper in [11] outlined the high-level structure of a CAD system
for reconfigurable array architectures, but it did not address how the modeling, simulation, and
evaluation could actually be performed and no experimental results were presented. The work­
bench in [12] aimed at reconfiguration issues only. It did not consider fault diagnosis issues,
and adopted a concurrent process hardware description language (CPHDL) for reconfigurable
architectures. However, it did provide a translator to allow direct translation of the generated
architectures into VHDL descriptions. The CAD systems in [13, 14, 15] are not specifically for
defect-tolerant architectures, while the CAD system in [16] is for the conceptual design of VLSI
systems. The VAR system presented in this chapter can fill this void. Kung [17] presented an
array compiler for CAD of array processors which can be divided into three levels: the array level, the processor level, and the realization level. The VAR system focuses on design and analysis issues at the processor level.

2.2. VAR Organization

This section overviews the VAR organization and problem-solving approach. The system configuration of the VAR system is shown in Figure 2.1 [18]. The module of array design with VHDL will be described in Chapter 3. The functions of other modules are specified as follows.

A. Kernel and VHDL Support Environment (K&VSE)

K&VSE is the shell of the VAR system. It supervises, schedules, and coordinates the tasks in VAR. In Figure 2.1 a dashed line between K&VSE and a module implies that there is an interaction involving commands and responses between these two modules. The Intermetrics Standard VHDL 1076 Support Environment [19], which is provided by the U. S. Air Force Wright Aeronautical Laboratories, executed on a Sun 3/260 workstation is the basis of VSE. VSE consists of the Design Database and four software packages: Analyzer, Simulator, VHDL Library System, and Design Library Manager. Simulator consists of Module Generator, Build, Sim, and Report Generator [19].

B. Design for Diagnosability (DFD) and Design for Reconfigurability (DFR)

In order to conquer the increasing testing complexity of VLSI/WSI architectures, DFD is an indispensable step toward easily diagnosable design. The measure of diagnosability can be expressed in terms of controllability and observability. Faced with the low yield problem of
VLSI/WSI architectures, DFR offers promise. It deals with the deployment and utilization of redundancy in an architecture. The measure of DFR can be expressed in term of reconfigurability. Reconfigurability determines how easily a defective architecture can be repaired through spare allocation and fault replacement. Before a VHDL-based architecture is simulated to assess its functionality, it will be measured for diagnosability and reconfigurability to ensure proper design of the architecture.
C. Fault Injection (FI)

FI handles how faults at different abstract levels and components (PEs, switches, or links) can be generated and represented in VHDL. Based on a fault model, the FI module generates and injects faults into an architecture. Faults can be generated by two types of defects, random defects and clustered defects. A random defect affects a wire or a single component. A clustered defect affects several components in an area. Usually, production yield loss is largely due to random defects [20].

D. Test Generation (TG)

A VHDL-based TG program generates test patterns for the fault diagnosis module. The TG program generates test patterns from the VHDL description of a circuit directly [21].

E. Fault Diagnosis Module (FDM)

In a reconfigurable system, replaceable faulty elements must be first identified before a repair action can be undertaken. FDM will execute a fault diagnosis algorithm specific to an architecture. Three fault diagnosis algorithms in Chapters 4 and 5 [22, 23, 24] are used here and the trade-offs can be compared via simulation.

F. Reconfiguration Module (RM)

RM coordinates the following three events: the execution of a reconfiguration program, the execution of a switching mechanism transformation program, and the VHDL simulation of target array generation and functionality verification. It will first execute the reconfiguration program to switch out faulty elements. There are two reconfiguration techniques, the redundancy approach and the degradation approach. In the redundancy approach, a system is built with some of its components dedicated as spare elements. These spare elements are used
to replace faulty elements in the reconfigurable system. In the degradation approach, all elements in a system are treated in a uniform way. No elements are dedicated as spare elements. The goal here is to derive a fault-free subsystem from the faulty system under certain limitations such as switching capabilities and the minimum dimension of the final system [25]. The reconfiguration algorithms in [25] which are based on the degradation approach are employed. Depending on the switching mechanisms, the algorithms reconfigure an array under three constraints: (1) row and column bypass (algorithm BB), (2) row bypass and column rerouting (algorithm BR), and (3) row and column rerouting (algorithm RR). The efficiency of the algorithms and the complexity of the switching mechanism transformation can be observed through simulation. For each reconfiguration algorithm, the corresponding switching mechanism may be different. Actual reconfiguration via VHDL simulation is used to verify the correctness of the reconfiguration algorithms and the switching mechanism transformation algorithms.

*FDM* and *RM* are fully integrated and in cooperation with each other in VAR to determine a fault-free configuration. A reconfiguration scheme used by *RM* determines a faulty diagnosis scheme to match it, and vice versa. For instance, if a reconfiguration scheme used by *RM* is based on a row/column replacement strategy, then *FDM* may want to choose a fault diagnosis scheme that stops performing diagnosis of a row/column if a faulty PE has been found on that row/column. If a reconfiguration scheme has a restriction on the size of the fault-free array, then *FDM* may want to choose a fault diagnosis scheme that stops diagnosing if the number of PE faults exceeds the fault tolerance capacity of the array. If a reconfiguration scheme is required to use the bypass link in a PE, then *FDM* may want to choose a fault diagnosis scheme that needs to determine not only if the function block of the PE is faulty, but also if the bypass link is faulty or not. On the other hand, depending on the fault distributions (*random faults* or *clustered faults*), *RM* can select an appropriate reconfiguration scheme. Some reconfiguration
schemes may perform better with the random fault distribution, while others may be good at handling clustered faults. The purpose of this interaction between FDM and RM is to minimize the diagnosis and reconfiguration time, to have high fault coverage, and to attain good PE utilization.

G. Evaluation Module (EM)

EM evaluates redundancy overhead, time complexity of fault diagnosis and reconfiguration algorithms, fault coverage, yield, delay, performance, and cost. The parameters that may affect yield are area overhead, diagnosability, reconfigurability, and the efficiency of diagnosis and reconfiguration algorithms. Overhead due to diagnosis and reconfiguration results in long wire delay and performance penalty. The correlation of these parameters is studied by EM. Variations in the design using different fault diagnosis algorithms, reconfiguration algorithms, and reconfigurable architectures are also evaluated based on the above parameters.

H. Optimization Module (OM)

Based on the evaluation results, OM modifies an architecture to enhance yield, performance, overhead reduction, and diagnosis and reconfiguration efficiency. The optimized architecture needs to be evaluated by EM again. This procedure may need to iterate several times before a cost-effective solution is obtained.

I. Synthesis Module (SM)

The ultimate goal of the VAR system is to generate a complete layout of a reconfigurable architecture directly from its VHDL description. SM can interface with a VHDL-based synthesis tool to translate the architecture from the behavioral description to a structural description and then from the structural description into a complete layout.
2.3. Summary

We have presented an integrated computer-aided design environment, the VAR (VHDL-based Array Reconfiguration) system, for the tasks of design, diagnosis, reconfiguration, simulation, and evaluation in a parallel architecture modeled by VHDL. VAR allows the designers to study and evaluate fault diagnosis and reconfiguration algorithms by inserting faults, which are generated based on manufacturing yield data, into an architecture and then locating the faulty PEs as well as simulating the reconfiguration process. Thus, VAR can assist the designers in evaluating different combinations of fault patterns, diagnosis algorithms, reconfiguration algorithms, and reconfigurable architectures through a complete set of figures of merit with aim at architectural improvements. Extensive simulation and evaluation will be performed in Chapter 6 to demonstrate and support the effectiveness of VAR.
CHAPTER 3.

ARRAY MODEL AND VHDL DESCRIPTION

3.1. Introduction

In this chapter, we present an array model. Based on this model, the overall system, two PE structures, and a switch structure are designed. A systolic array is used as an example to illustrate the modeling process for defect-tolerant architectures using VHDL. The descriptions of the array model, system design, PE design, switch design, and the comparison with an existing modeling method are detailed in the following sections.

3.2. Array Model and Its Implication

A general PE-switch lattice model is shown in Figure 3.1 [26,27], where a square represents a PE and a horizontal (vertical) line represents a communication link. A row or column consisting of PEs and links is a PE track; otherwise, it is called a link track if it consists of links only. A switch may exist at the intersection of a link track and a PE track, and such a switch is called a cardinal switch. A switch may also be located at the intersection of a horizontal link track and a vertical link track, and such a switch is called a corner switch. These two types of switches have the same implementation. A type-1 reconfigurable array contains only cardinal switches and no switches at the array boundaries. A type-2 reconfigurable array contains both cardinal and corner switches. The type-1 array uses fewer switches but with
less reconfiguration capability than the type-2 array. The notation \(<M, N, T_h, T_v, S, U, V>\) is used to represent and characterize such an array model, where \(M \times N\) is the size of the host array; \(T_h\) and \(T_v\) are the numbers of horizontal and vertical link tracks between two adjacent PE tracks respectively; \(S\) indicates a type-1 (\(S = 1\)) or a type-2 (\(S = 2\)) array; and \(U \times V\) is the size of the target array. The array I/O ports are assumed at the array boundaries: row 0, row \(M+1\), column 0 and column \(N+1\). For a reconfigurable array based on the degradation approach, \(U = V = 0\) is set if there is no restriction on the size of the target array. For the ease of discussion, some necessary definitions and terminologies originating from this array model are introduced next.
Given the array model $<M, N, T_h, T_v, S, U, V>$, a *non-defect-tolerant array* is a $U \times V$ array with no redundant elements such as extra PEs, switches, or links; i.e., it is the original architecture for a particular application. A *host array* is an $M \times N$ reconfigurable PE-switch array which may contain faulty elements after manufacturing. A *logic array* is a host array with the faulty switches and links switched out using the method in Chapter 4 [26]. Note that a logic array may still contain faulty PEs. A *target array* is a $U \times V$ reconfigurable PE-switch array that contains no faulty elements and is obtained by replacing faulty elements through reconfiguration. Although a non-defect-tolerant array and the corresponding target array are functionally equivalent, the performance of the latter might be worse than the former due to the switches and longer wires.

A *logic matrix* is a $U \times V$ matrix in which an element $(r, c)$ in row $i$ and column $j$ represents PE($i$, $j$) of a target array and is located in row $r$ and column $c$ of the host array. An *augmented logic matrix* is a $(U+2) \times (V+2)$ expanded logic matrix with the locations of I/O ports included. An *error matrix* is a $(F+1) \times 1$ matrix which contains the size of a host array and each element $(r, c)$ represents a faulty PE located in row $r$ and column $c$ of the host array, where $F$ is the number of faulty PEs. A *plain-switch matrix* is a matrix for controlling the states of switches in the vertical (or horizontal) link tracks of a host array. A *mix-switch matrix* is a matrix for controlling the states of switches in the vertical (or horizontal) PE tracks of a host array. A *PE matrix* is an $M \times N$ matrix for controlling the states of PEs in a host array. A *characteristic matrix* is a matrix in which each element represents the state of either a PE or a switch in a reconfigurable array. The size of a characteristic matrix is $(T_h(M-1)+M) \times (T_v(N-1)+N)$ for a type-1 array and is $(T_h(M+1)+M) \times (T_v(N+1)+N)$ for a type-2 array. A characteristic matrix characterizes the status of a host array before reconfiguration and the status of a target array after reconfiguration. It can be decomposed into a plain-switch matrix, a mix-
switch matrix, and a PE matrix. The classification used in the characteristic matrix is shown in Figure 3.2, where each leaf node is a possible state of a PE or switch. The classification concept is related to the work by Pradhan [28].

3.3. Array Design and Its VHDL Description

3.3.1. System Design

A reconfigurable array \(<M, N, T_h, T_v, S, U, V>\) is used to demonstrate the approach of top-down design and bottom-up implementation. Figure 3.3 shows the hierarchical VHDL description of the reconfigurable array. It is a modular structure and consequently is easy for complexity management, design modification, or implementation change with different components by rebinding the design via the configuration declaration in VHDL [29]. The description of the array is divided into two types of blocks, switch blocks and PE-switch blocks. A

![Diagram](image)

*Figure 3.2. Classification used in the characteristic matrix.*
link track in the array of Figure 3.1 is a switch block which consists of switches and links, while a PE track is a PE-switch block which consists of PEs as well as switches and links. In Figure 3.3, only the leaf nodes are modeled with VHDL behavioral descriptions and all higher level nodes are modeled with VHDL structural descriptions. The array is described in a generic way. The host and target array dimensions ($M \times N$ and $U \times V$), the numbers of horizontal and vertical link tracks ($T_h$, $T_v$), the array type ($S$), the clock cycle times of PEs, switches, and control lines ($PE_{cycle\_time}$, $SW_{cycle\_time}$, $Ctr_{cycle\_time}$), and the delays of subcomponents ($Multiplier_{delay}$, $Adder_{delay}$, etc.) are described by a VHDL generic statement to allow the designers to specify these parameters and hence, to facilitate design flexibility.

In order to simulate such a reconfigurable array, a VHDL test bench environment is created to facilitate test and simulation. Figure 3.4 demonstrates the VHDL test bench environment for reconfigurable arrays. The VHDL test bench includes two components: the generator ($Systolic_{gen}$) and the reactor ($Systolic_{array}$). When the VHDL test bench is simulated, $Systolic_{gen}$ will provide stimulus to $Systolic_{array}$, and $Systolic_{array}$ will respond to this stimulus and will send results back to $Systolic_{gen}$ [30]. By using the test bench environment,
we can experiment with different design alternatives of Systolic_array and various types of Systolic_gen. For instance, this environment can be used by FDM for the simulation of fault diagnosis process. In this case, the Systolic_gen supplies the test patterns, while the Systolic_array responds with the test results. In Chapter 6, the test bench environment will be used by RM to simulate the generation and the functionality verification of the target array.

The VHDL description of the test bench environment is depicted as follows. Figure 3.5 shows the VHDL entity declaration with a generic statement and two local component declaration statements of the test bench environment. The generic statement includes the above parameters with default values, which can also be supplied during simulation without reanalyzing the VHDL codes. The non-basic data types are defined in the package declaration to facilitate resource sharing and modification. The data type of port with the prefix "Resol_" represents that the port has multiple sources and is defined as a resolved signal that has an associated resolution function to determine the value of the resolved signal [31]. Note that those time-related parameters have the data type of Positive or Natural. This is because the VHDL simulator restricts the generic parameters to be of basic data types only. Therefore, these parameters need to be converted into the data type of Time with the time unit ns during component instantiation.
entity Systolic_test is
generic(M, N : Positive;
    U, V : Positive;
    Th : Natural := 1;
    Tv : Natural := 1;
    S : Natural := 2;
    PE_cycle_time : Positive := 200;
    Ctr_cycle_time : Positive := 20;
    SW_cycle_time : Positive := 20;
    Multiplier_delay : Natural := 80;
    Adder_delay : Natural := 20;
    Demux12_delay : Natural := 10;
    Mux31_delay : Natural := 10;
    Mux21_delay : Natural := 10;
    Data_reg_delay : Natural := 10;
    Ctr_reg_delay : Natural := 10;
    NAND3_delay : Natural := 5;
    SW_comm_delay : Natural := 10;
    SW_reg_delay : Natural := 10);
end Systolic_test;

component Systolic_gen
    generic(M, N : Positive;
    U, V : Positive;
    Th : Natural;
    Tv : Natural;
    S : Natural;
    PE_cycle_time : Positive;
    Ctr_cycle_time : Positive;
    SW_cycle_time : Positive);
port(SW_clk_a : out Bit;
    SW_clk_a : out Bit;
    CR_in_a : out SW_reg_vector(1 to N+1);
    CR_out_a : in SW_reg_vector(1 to N+1);
    PE_clk_a : out Bit;
    Ctr_clk_a : out Bit;
    CR_in_ae : out SW_reg_vector(1 to N);
    CR_out_ae : in SW_reg_vector(1 to N);
    DM_clk_a : out Bit;
    X_in_a : inout Resol_word_vector(1 to M);
    Y_in_a : inout Resol_word_vector(1 to N);

Figure 3.5. VHDL entity and component declarations of the test bench environment.
The component *Systolic_array* is the VHDL description of the example array, which is hierarchical and modular in the manner described by Figure 3.3. Various component design options can be easily experimented. The component *Systolic_gen* is used to:
(1) generate the PE, switch, and control clocks,

(2) read and transmit the following data to Systolic\_array: PE and switch control data, and input data for multiplication, and

(3) receive the multiplication results from Systolic\_array.

The above events are synchronized in Systolic\_gen by adapting the synchronization mechanism to the host arrays and target arrays of various sizes, and different clock cycle times. Since this is an architecture-level design system, a single delay time is used to capture all possible inherent delay inside a subcomponent, such as propagation delay and capacitive charging and discharging delay, etc.

3.3.2. PE Design

Two types of PEs (PE\(_a\) and PE\(_b\)) are designed to illustrate how to incorporate and evaluate different component implementations. The structure of PE\(_a\) is shown in Figure 3.6(a) [26]. The I/O relationships of PE\(_a\) are \(Y_{\text{out}} \leftarrow Y_{\text{in}}\) and \(X_{\text{out}} \leftarrow X_{\text{in}} + W \times Y_{\text{in}}\). It has a demultiplexer (D), a multiplexer (M), three 16-bit data registers (W, Rx, and Ry), a multiplier (Mpy), an adder (Add) and a 2-bit control register (C\(_{dn}\)). The size of each data register can be easily modified in the VHDL package declaration. The control register is controlled by a control clock \(CTR_{clk}\) and a control line \(DM_{ctrl}\). When \(DM_{ctrl} = \'0\)', the control registers of the PEs in the same row of the array form a scan path through \(DM_{in}\) and \(DM_{out}\) which can scan in the PE control data. The demultiplexer and the multiplexer are controlled by the output of the control register. If \(C_{dn}(0) = \'1\)', the corresponding PE is bypassed. Each data register has two clock inputs (\(PE_{clk0}\) and \(PE_{clk1}\)) controlled by the value of \(C_{dn}(1)\). \(PE_{clk0}\) is the normal clock for an active PE when \(C_{dn}(1) = \'0\'). \(PE_{clk1}\) has a shorter clock cycle and is used when \(C_{dn}(1) = \'1\'). It is the clock
Figure 3.6. Two PE structures.

for a data register when the corresponding PE is bypassed. A data register functions as a delay register when the corresponding PE is being bypassed. The other PE design (PEb) (Figure
3.6(b) has a similar configuration to $PE_a$ with the addition of a bypass link to each data path. $PE_b$ needs only one PE clock $PE_{clk}$ for each data register. The VHDL entity declaration of $PE_b$ is shown in Figure 3.7 which shows the generic parameters and port definitions corresponding to the $PE_b$ design in Figure 3.6(b). The operation of each register in either PE is triggered by the falling edge of a single-phase clock. The single-phase clock scheme is adopted to focus on system design issues. Although MOS circuits are typically driven by a two-phase clock to avoid the clock skew problem, it will always be possible to translate directly the resulting circuits to corresponding two-phase MOS implementation [32].

3.3.3. Switch Design

The switch structure is shown in Figure 3.8 [26]. It has four I/O ports ($SW_e$, $SW_w$, $SW_s$, and $SW_n$) and consists of a switch communication box ($SW_{comm}$) and a 4-bit switch control

entity $PE_b$ is
generic(Multiplier_delay : Time;
    Adder_delay : Time;
    Demux12_delay : Time;
    Mux31_delay : Time;
    Mux21_delay : Time;
    Data_reg_delay : Time;
    Ctr_reg_delay : Time;
    NAND3_delay : Time);
port(PE_clk, Ctr_clk : in Bit;
    DM3tr : in Bit;
    X_in, Y_in : in Word;
    DM_in : in Ctr_reg_size;
    X_out, Y_out : out Word;
    DM_out : out Ctr_reg_size);
end PE;

Figure 3.7. VHDL entity declaration of $PE_b$. 
register ($SW_{reg}$). The switch control register has a clock line $SW_{clk}$, a control line $SW_{ctr}$, and a control register input (output) line $CR_{in}$ ($CR_{out}$). When $SW_{ctr} = '0'$, the switch control registers in the same column of the array form a scan path through $CR_{in}$ and $CR_{out}$ which can scan in the switch control data. The switch communication box has three types of connection patterns with four possible states for each type as shown in Figure 3.9. That is, there are 12 possible states which are controlled by the output ($SC_{ctr}$) of the switch control register. Although not all of the states are used in this example array, all the possible states are shown for completeness. The VHDL entity declaration of the switch is shown in Figure 3.10, where each port is bidirectional with the data type of inout. The switch design is more efficient than the design in [33] in terms of the reconfiguration time of switches and design complexity.
3.4. Comparison with an Existing Modeling Method

The Object-Oriented Design of Reliable/Reconfigurable Architectures (OODRA) workbench was targeted at the design and analysis of concurrent process message-passing based, parallel reconfigurable architectures [12]. It used a concurrent process model for the description of application specific reconfigurable parallel architectures. This concurrent process model has been encapsulated in CPHDL. Although VHDL and CPHDL both have the same features that can describe concurrency and structural reconfiguration, CPHDL was chosen for OODRA instead of VHDL. In [12], it argues that VHDL is too verbose to be easily captured in a simulation model targeted at application specific parallel architectures. However, eventually a translator is still needed to convert the CPHDL descriptions into the VHDL descriptions and might face the same problem in the translator design. Based on our experience and the fact of VHDL being an IEEE standard, this problem can be overcome. When we talk about tool support and
entity SW is
  generic(SW_comm_delay : Time;  
          SW_reg_delay : Time);
  port(SW_clk : in Bit; 
       SW_ctr : in Bit; 
       CR_in : in SW_reg_size; 
       CR_out : out SW_reg_size; 
       SW_w : inout Word; 
       SW_e : inout Word; 
       SW_n : inout Word; 
       SW_s : inout Word);
end SW;

Figure 3.10. VHDL entity declaration of the switch.

The interface with other tools, VHDL is the choice for the description of reconfigurable architectures. Other existing models of hardware execution and their associated hardware description languages are not targeted toward describing reconfigurable architectures, and are reviewed in [12].

Now we will compare the differences of using CPHDL in OODRA and VHDL in VAR to describe reconfigurable architectures. There are three basic elements in a CPHDL-based reconfigurable architecture description: process abstractions, switch abstractions, and link abstractions [12]. A reconfigurable architecture is composed of multiple processors implementing processes interconnected by links and reconfigured by switch mechanisms [12]. The VHDL description of a reconfigurable architecture is based on the hierarchical structure in Figure 3.3. One extra level of hierarchy which consists of the switch block and the PE-switch block is added, so we only deal with one dimension of interconnections instead of two dimensions of interconnections as in the CPHDL-based architectural description. This has the advantage of making architectural description concise and clear. In OODRA, both the host array and the non-defect-tolerant array need to be described by CPHDL. The description of the non-defect-
tolerant array is used as a template for the purpose of mapping the host array with multiple faults into a working system (target array) [12]. The link and channel structures are static; i.e., once instantiated, they can not be physically rerouted. In VAR, only the host array is described in VHDL. The target array is obtained through actual reconfiguration by using the PE and switch control data to convert the host array into the target array. The interconnection patterns are dynamic in our approach; i.e., they can be rerouted by using different PE and switch control data. Therefore, by describing the host array in VHDL, we can reconstruct and simulate various target arrays based on different fault patterns. The above discussion outlines the basic differences of the modeling methods between CPHDL in OODRA and VHDL in VAR. This comparison demonstrates the effectiveness and flexibility of the VHDL modeling method in VAR.

3.5. Summary

In this chapter, we have discussed the modeling process for a defect-tolerant two-dimensional array. VHDL has been used to model and describe such an array. In Chapter 6, the VHDL simulation of this array will be performed and the experimental results will be analyzed to demonstrate the effectiveness of our modeling method.
CHAPTER 4.

FAULT DIAGNOSIS

Two fault diagnosis approaches are presented in this chapter. The parallel partition approach has the advantage of testing speed-up with very little hardware overhead, while the self-comparison approach has the advantage of better testing speed-up with a little higher hardware overhead. In both approaches, array architectures are used to illustrate their principles. These two approaches are also applicable to other parallel architectures.

4.1. Parallel Partition Approach

4.4.1. Introduction

Although the ultimate goal of the development of VLSI technologies is toward Wafer Scale Integration, the yield problem is a major concern. Design for reconfiguration is currently one promising way for yield enhancement. This method incorporates redundancy into a chip or whole wafer. Faulty elements can then be replaced by spare elements. But before reconfiguration can be performed, locations of faulty elements must be identified. The number of processing elements (PEs) in an array can be large, thus testing one PE at a time will be very time consuming. Since all the PEs in an array are identical, each PE has the same set of test patterns which can be generated by traditional test generation programs. To speed up test appli-
cation, we add some extra hardware to each PE and explore various characteristics of an architecture such as regularity, locality, and concurrency. The yield can then be improved through a fault replacement or reconfiguration process [2].

VLSI and WSI arrays have found applications in real-time and high throughput signal and image processing. The testing of such array architectures has been previously studied. Many existing testing methods for array architectures such as iterative logic arrays or cellular automata are for fault detection only [34, 35, 36, 37]. The idea of partitioning also appears in [38] but with different contexts. Its objective is to partition a system into subsets that can be tested concurrently. It again addresses the fault detection of a system only. Most diagnosis methods that address fault location for arrays are comparison-based or voting schemes [4, 39]. Both the comparison and voting methods have difficulty in diagnosing multiple faults, low locatability, and extensive hardware overhead. A fault-tolerant two-dimensional systolic array was designed in [7]. Although the array is designed to be testable, the hardware overhead is high. Ramamoorthy [40] is the first researcher to use graph representation for the fault diagnosis process. He used a connectivity matrix to find the precedence partition [40] in the testing of multiprocessor systems. Test points were added to increase the observability of systems under test. His approach is not suitable for large systems where the connectivity matrix can become very large. The use of test points also means extra pins which are limited resources in VLSI [41].

In this section, we present the parallel partition approach which can speed up the detection and location of multiple PE-faults in VLSI/WSI array architectures. The parallel partition approach divides an array under test into disjoint diagnosis blocks in which PEs in the same diagnosis block can be tested concurrently. This approach reduces the complexity of fault location of an array to that of fault detection of a single PE. Three types of PEs and one type of
switches are designed to show how to apply this approach. The bypass register links (BRLs) in each PE are used to construct testing paths so that test patterns can be applied and test results can be observed from boundary I/O ports. Depending on the type of PE, little or no hardware is added for the purpose of diagnosis, and the added hardware will additionally improve reconfigurability by providing an extra track for reconfiguration. This is significant because high hardware overhead increases the area of the array which means more faults and low yield.

The remaining subsections are organized in the following way. Some terminologies and definitions are first described. The parallel partition approach is illustrated in Section 4.1.3. The diagnosis of an example rectangular array using this approach is then illustrated. In Section 4.1.4, the designs of PEs and switches in example systolic arrays are detailed. In Section 4.1.5, the diagnosis algorithms for PE, switch, and link faults are depicted. The parallel partition approach is then extended to other parallel architectures in Section 4.1.6 and followed with a summary.

4.1.2. Terminologies and Definitions

The PE-switch lattice model described in Chapter 3 is the array model used to illustrate the parallel partition approach. Some related terminologies have been defined in Chapter 3. A lattice array is a host array with all its PEs being bypassed. In order to facilitate the testing and diagnosis process, a matrix is used to represent the status of each element in a host array. The array status matrix (ASM) is a $P \times Q$ matrix in which each entry represents the status of a PE or switch, and two adjacent entries of ASM indirectly represent the status of the link between the two corresponding elements in a host array, where $P$ represents the total number of horizontal PE and link tracks and $Q$ represents the total number of vertical PE and link tracks.
\( P \times Q \) is \( (T_h(M-1)+M) \times (T_v(N-1)+N) \) for a type-1 array and is \( (T_h(M+1)+M) \times (T_v(N+1)+N) \) for a type-2 array. A value 0 at \( ASM[i,j] \) signifies that the corresponding PE or switch is fault-free and all its adjacent links are fault-free. A value 1 at \( ASM[i,j] \) signifies that the corresponding PE or switch has a fatal fault and all its adjacent links are fault-free. A value 2 at \( ASM[i,j] \) corresponding to a PE signifies that the PE is faulty, but both its bypass (register) links and all its adjacent links are fault-free. A value 3 at \( ASM[i,j] \) signifies that the corresponding PE or switch is fault-free and at least one of its adjacent links is faulty. A value 4 at \( ASM[i,j] \) signifies that the corresponding PE or switch has a fatal fault and at least one of its adjacent links is faulty. A value 5 at \( ASM[i,j] \) corresponding to a PE signifies that the PE is faulty but its bypass (register) links are fault-free, and at least one of its adjacent links is faulty. Therefore, two adjacent entries in \( ASM \) with values either 3, 4 or 5 imply that the link between the two corresponding elements is faulty. This faulty link representation scheme has an advantage of using a small \( ASM \) to represent a faulty array with a large number of links and relatively few link faults. But the disadvantage is that a fault-free link can be labeled as faulty, although it is rare. A link status matrix is used to compensate for this if such ambiguity exists.

A link status matrix is a \( E \times 1 \) matrix such that each entry which contains four indices \((r_1, c_1, r_2, c_2)\) represents a faulty link, where \((r_1, c_1)\) and \((r_2, c_2)\) are the locations of switches and/or PEs at the two ends of the link in a host array, and \( E \) is the number of faulty links.

A level-n row-originated data path represented by \( RP_{k_1k_2k_3\ldots k_n} \) is a data path originating at the input port of row \( k_1 \), bending at column \( k_2 \), bending at row \( k_3 \), ... , and terminating at the output port of row (column) \( k_n \) in a lattice array by setting proper switches if \( n \) is an odd (even) number. A level-n column-originated data path represented by \( CP_{k_1k_2k_3\ldots k_n} \) is a data path
originating at the input port of column $k_1$, bending at row $k_2$, bending at column $k_3$, ..., and terminating at the output port of column (row) $k_n$ in a lattice array by setting proper switches if $n$ is an odd (even) number. A level-$n$ row operator (column operator) represented by $RO_{k_1k_2 \cdots k_n}$ ($CO_{k_1k_2 \cdots k_n}$) is an operator that is used to initiate the testing of the data path $RP_{k_1k_2 \cdots k_n}$ ($CP_{k_1k_2 \cdots k_n}$) by applying test vectors at the input port and observes test responses at the output port of the data path. A value 0 (1) is used to represent the test result of a data path if the test result is correct (incorrect) after applying a level-$n$ operator. A row status vector (RSV) is a $P \times 1$ vector in which $RSV[i]$ records the status of $RP_i$ in a lattice array after applying the level-1 row operator $RO_i$. A column status vector (CSV) is a $1 \times Q$ vector in which $CSV[j]$ records the status of $CP_j$ in a lattice array after applying the level-1 column operator $CO_j$. A row status matrix (RSM) is a $P \times Q$ matrix where $RSM[i,j]$ represents the status of $RP_{ij}$ in a lattice array after applying the level-2 row operator $RO_{ij}$. A column status matrix (CSM) is a $P \times Q$ matrix where $CSM[i,j]$ represents the status of $CP_{ji}$ in a lattice array after applying the level-2 column operator $CO_{ji}$.

4.1.3. Parallel Partition

An example 5 x 6 logic array in Figure 4.1 is used to demonstrate how the parallel partition approach works. In Figure 4.1, each PE is represented as $PE(i,j)$, where $i$ represents the row index and $j$ represents the column index. A general PE model is presented in Figure 4.2, where $M_h$ and $M_v$ are multiplexers. The links associated with the registers $R_h$ and $R_v$ can become BRLs by controlling the corresponding multiplexers. Each data path in this PE model has a BRL to enhance testability. A BRL can also provide an extra track for reconfiguration and avoid a long connection by using a buffer register. If a data path already has a BRL, no
extra BRL is required for that data path. That is, no extra hardware (a register and a multiplexer) is required for that data path. Usually a PE in systolic arrays already has BRLs in most of its data paths. In Section 4.1.4, we will explain this and demonstrate how this model suits typical PE structures in systolic arrays. The BRLs make it possible to directly apply test patterns to a PE and observe test responses from array I/O ports. For example, to test PE(3,2) in Figure 4.1, test inputs and test responses can pass the other PEs in the same row and column through the BRLs of these PEs. A PE may be in one of two operation modes, the normal mode and the bypass mode, during the diagnosis process. In the normal mode, a PE performs regular operations on input data and passes test results to neighboring PEs. PEs under test are operating in this mode. In the bypass mode, input data are routed through a PE via internal BRLs without being processed. PEs not under test are in the bypass mode and provide scan paths for
In order to reduce the complexity of test application, a logic array is partitioned into dis-joint diagnosis blocks. A *diagnosis block* (DB) is a set of PEs whose output data paths to the array output ports do not share any interconnection link. A *maximal diagnosis block* (MDB) is a DB that is not a subset of any other DB. To derive MDBs for a rectangular logic array, we first obtain the diagonal groups of the logic array. A *diagonal group* is a set of PEs such that two PEs, PE\((i_1, j_1)\) and PE\((i_2, j_2)\), are in the same group \(g_k\) if and only if \(i_1+j_1 = i_2+j_2\) where \(k\) is defined as \(i_1+j_1-1\). For example, three PEs in Figure 4.1, PE\((1,3)\), PE\((2,2)\) and PE\((3,1)\), are in the same diagonal group \(g_3\). The parallel dotted lines in Figure 4.1 show the diagonal groups of the logic array. PEs in the same group are guaranteed to share no links among their output paths since they do not lie on the same row or column. This can be easily seen from the definition of a diagonal group. An MDB may consist of one or more diagonal groups if PEs in these groups satisfy the requirement of output data path independence. The
two groups in Figure 4.1, \( g_3 = \{\text{PE}(1,3), \text{PE}(2,2), \text{PE}(3,1)\} \) and \( g_9 = \{\text{PE}(4,6), \text{PE}(5,5)\} \), form an MDB because their output data paths are disjoint, and these PEs can be diagnosed in parallel.

The parallel partition problem is shown to be equivalent to a generalized *Eight Queens* problem as follows. A generalized Eight Queens problem is the placement of as many queens as possible on an \( n \times n \) chessboard so that no queens can attack any other queens. Since a queen attacks another queen on the same row, column, or its diagonals, clearly at most, \( n \) queens can be placed on a chessboard [42]. The array partition problem is similar to the Queens problem with output data paths corresponding to rows, columns, or diagonals of a chessboard. In Figure 4.1, there are row and column output data paths and no diagonal output data paths. But a hexagonal logic array has row, column, and diagonal output paths. The difference between the parallel partition problem here and the Queens problem is that we are not interested in finding the number of partitions. Instead, we are interested in finding a partition with a minimum number of MDBs. Finding a minimum set of MDBs can be solved in polynomial time as shown at the end of this section. Based on the above discussion, we have the following lemma.

**Lemma 4.1:** The parallel partition problem is equivalent to a generalized Eight Queens problem.

\[ \square \]

Before we present the parallel partition algorithm, the following properties are stated first.

**Lemma 4.2:** The number of PEs in group \( g_k \) for an \( m \times n \) logic array is:

\[
\begin{align*}
  k & \quad \text{if } k < \min(m,n) \\
  m+n-k & \quad \text{if } k > \max(m,n) \\
  \min(m,n) & \quad \text{otherwise}
\end{align*}
\]
**Proof:** Without loss of generality, assume \( m < n \). A PE\((i,j)\) is in \( g_k \) if \( k = i+j-1 \) (1 \( \leq \) \( i \leq m \), 1 \( \leq \) \( j \leq n \). Hence \( k+1-n \leq i \leq k \) with 1 \( \leq \) \( j \leq n \). For Case (1), \( k < m = \min (m,n) \), which implies 1 \( \leq \) \( i \leq k \). That means PEs in group \( g_k \) have row indices from 1 to \( k \) which is equivalent to saying that there are \( k \) PEs in this group. For Case (2), \( k > n = \max (m,n) \) which implies \( k+1-n \leq i \leq m \). So the number of PEs in group \( g_k \) is \( m+n-k \). For the last case, \( \min (m,n) = m \leq k \leq n = \max (m,n) \), we have 1 \( \leq \) \( i \leq m \) which means the number of PEs in group \( g_k \) is \( m \).

\( \Box \)

**Lemma 4.3:** There are \( m+n-1 \) diagonal groups in an \( m \times n \) logic array.

**Proof:** Since 1 \( \leq \) \( i \leq m \) and 1 \( \leq \) \( j \leq n \) for a PE\((i,j)\) in this array, it implies 1 \( \leq \) \( i+j-1 \leq m+n-1 \). PE\((i,j)\) is in group \( g_k \) \((k = i+j-1)\) based on the definition of diagonal groups. Therefore there are \( m+n-1 \) groups in an \( m \times n \) logic array. 

\( \Box \)

The following lemma is similar to the problem of placing maximal number of queens on an \( n \times n \) chessboard.

**Lemma 4.4:** In an \( m \times n \) logic array, the number of PEs in an MDB is \( \min (m,n) \) and the number of MDBs is \( \max (m,n) \).

**Proof:** Without loss of generality, assume \( m < n \). There are at most \( m \) PEs in an MDB; otherwise, two PEs will be in the same row. These \( m \) PEs can be in different columns since \( m \) \( < \) \( n \). These \( m \) PEs form an MDB and the number of MDBs is \( n = (m \times n)/m \).

\( \Box \)
Theorem 4.1: For a diagonal group $g_k$ in a DB $b$ of an $m \times n$ logic array,

1) If $k < \min(m, n)$, the group $g_l$ where $l = k + \max(m, n)$ can also be included in $b$ and $b$ is an MDB.

2) If $k > \max(m, n)$, the group $g_l$ where $l = k - \max(m, n)$ can also be included in $b$ and $b$ is an MDB.

3) If $\min(m, n) \leq k \leq \max(m, n)$, $b$ is an MDB which has only one group $g_k$.

Proof: Again assume $m < n$.

1) $k < m$ and $l = k + n$

We know from Lemma 4.4 that an MDB will have $m$ PEs. If the number of PEs in one group $g_k$ is less than $\min(m, n) = m$, another group $g_l$ ($l = k + n$) can be included to form an MDB. Let $i$ be the row index of a PE in $g_k$ and $j$ be the row index of a PE in $g_l$. From the proof of Lemma 4.2, $1 \leq i \leq k$ and $k+n+1-n = k+1 \leq j \leq m$. Since $n > m$, these PEs in $g_k$ and $g_l$ are in different rows and they form a DB $b$. From Lemma 4.2 group $g_k$ has $N_1 = k$ PEs and group $g_l$ has $N_2 = m+n-k-n$ PEs. $N_1 + N_2 = k+m+n-k-n = m$. Therefore the DB $b$ with groups $g_k$ and $g_l$ is an MDB.

2) $k > n$ and $l = k - n$

Following the similar reasoning in Case (1), it can be shown that PEs in $g_l$ and $g_k$ form a DB $b$. Group $g_k$ has $N_1 = m+n-k$ PEs and group $g_l$ has $N_2 = k-n$ PEs. $N_1 + N_2 = m+n-k+k-n = m$. So the DB $b$ is an MDB.

3) $m \leq k \leq n$

Group $g_k$ has $m$ PEs and hence the DB $b$ containing only one group $g_k$ is an MDB. \qed
Note that Case (1) and Case (2) in Theorem 4.1 actually address the same set of pairs of groups. The BRLs of PEs in the other MDBs are activated to establish I/O paths directly to PEs in an MDB under test. Since these output paths do not interfere with each other, we have the following corollary.

**Corollary 4.1:** PEs in one MDB of a logic array can be diagnosed concurrently by activating the BRLs of PEs in other MDBs. The test patterns of the logic array is that of a single PE.

**Proof:** From Theorem 4.1, the output paths of PEs in an MDB do not share links. By activating the BRLs of PEs in other MDBs, every PE in this MDB has its own output data paths to array output ports. We can then apply the same single PE test patterns to these PEs in parallel and observe the test results at the corresponding output ports concurrently. The locations of faulty PEs can be identified since the test result of each PE appears at a different output port.

Based on Theorem 4.1, the algorithm to find a set of MDBs for a rectangular logic array is shown in Figure 4.3. First all diagonal groups are derived, then MDBs are formed from these groups. The complexity of this algorithm is $O(mn)$. The detailed testing and diagnosis process for example systolic arrays is presented in Section 4.1.5.

**4.1.4. Design of Easily Testable PEs**

As mentioned before, the parallel partition approach reduces the complexity of fault diagnosis of an array to that of fault detection of a single PE. Three typical PEs for two-dimensional systolic arrays are designed to be easily testable and reconfigurable to implement and illustrate this approach. The switch design has been described in Chapter 3.
Algorithm *Parallel Partition*

*Begin*

/* get all diagonal groups */

1. For every PE\((i,j)\) in an \(m \times n\) logic array do
   
   \(k := i+j-1;\)
   
   \(g_k := g_k \cup \{\text{PE}(i,j)\};\)
   
   End-for;

/* get all maximal diagnosis blocks*/

2. For \(k := 1\) to \(\max(m,n)\) do
   
   If \(k < \min(m,n)\) Then
   
   \(l := k+\max(m,n);\)
   
   \(b_k := b_k \cup g_l;\)
   
   Else
   
   \(b_k := g_k;\)
   
   End-if;
   
   End-for.

*End.*

**Figure 4.3.** Parallel partition algorithm.

### 4.1.4.1. Example PE Configurations

Typical PEs used in systolic array architectures are inner product step processors (IPSPs). The main function of an IPSP is to perform an inner product step function, \(X \leftarrow X + W \times Y\). That is, the basic operations of an IPSP are multiplication and addition. Three typical PEs used in mesh or hexagonal systolic arrays are shown in Figure 4.4 [43]. Other types of PEs used in linear systolic arrays have similar configurations. In Figure 4.4, we see that all the data paths except one in a PE are actually BRLs. A non-BRL data path may have a BRL. This fact will become clear on the testable designs of PEs in the next subsection.
4.1.4.2. Detailed PE Designs

The easily testable designs of the three PEs are shown in Figure 4.5. For PEa (Figure 4.5(a)), the existing hardware includes a multiplier (Mpy), an adder (Add), three registers (W, \(R_x, R_y\)), a demultiplexer (D), a 1-bit control flip flop (\(C_d\)), a multiplexer (M), and a 1-bit control flip flop (\(C_m\)). The demultiplexer and the multiplexer are necessary for preloading and scanning a W value into each PE for normal operations. By controlling the demultiplexer and the multiplexer, the W register and the associate link can become a BRL for the horizontal data path by setting \(C_d = 0\) and \(C_m = 1\). Note that the vertical data path is already a BRL. As a result, no extra hardware is needed in PEa for the purpose of diagnosis. For PEb (Figure 4.5(b)), the only extra hardware needed is the use of a selector (S) instead of a multiplexer. A multiplexer would normally be used for scanning out the product of multiplication in normal operations. A selector instead of a multiplexer is used to make the multiplier and the adder easily testable and help to create a BRL. Therefore, the extra hardware for diagnosis is very insignificant. The selector is controlled by a 2-bit control flip flop (\(C_s\)) which has three states as shown in Figure 4.6. State SEL0 (\(C_s = "00"\)) is for normal operations. State SEL1 (\(C_s = "01"\)) is used to
Figure 4.5. Testable designs of the three PEs in Figure 4.4.
 preload a $W$ value. State $SEL_2 (C_s = "10")$ is used to make the test result of each PE observable. Note that the vertical data path is a BRL. As for $PE_e$ (Figure 4.5(c)), the only extra hardware needed is a multiplexer ($M$) and a 1-bit control flip flop ($C_m$) to form a BRL in the $Y$ data path by setting $C_m = '1'$. The $W$ and $X$ data paths are BRLs. Hence, the actual extra hardware for diagnosis is still very little, and in addition, it can improve reconfigurability.

For all three types of PEs, a testing clock is used to scan in test patterns and scan out test results during diagnosis, and a system clock is used to initiate normal operations of PEs. Both the multiplier and the adder in any PE can be purely combinational circuits. The adder can be a ripple carry adder which is C-testable [44]. The multiplier can also be designed as an array multiplier which is again C-testable [36,45]. The multiplier and the adder in a PE are designed to be tested individually. The faults in the rest of the PE which is part of internal data paths are handled during the control register testing and the switch and link testing. So a PE may be tested with a constant number of test patterns depending on its actual implementation. Previous studies showed that a ripple carry adder can be tested with 8 test patterns [44] while a modified carry-save array multiplier can be tested with 16 test patterns [45].
4.1.5. Testing and Diagnosis

The $PE_a$-based systolic array is used in the following discussion to illustrate the testing and diagnosis process. There are two types of testing: on-line and off-line testing. The types of faults detected are operational and production faults. The algorithms proposed here are for off-line production faults testing. The fault model is presented first. Based on this fault model, the testing and diagnosis process consists of three phases: (1) testing of control registers in PEs and switches, (2) switch and link testing, and (3) diagnosis of multiple PE-faults. The overall testing and diagnosis process is illustrated in Figure 4.7. Details of each functional module in Figure 4.7 are described in Sections 4.1.5.2, 4.1.5.3, and 4.1.5.4.

4.1.5.1. Fault Model

All multiple PE, switch, and link faults are considered here. The fault model is similar to the one used by Kim and Reddy [7]. Assume that faults in a PE functional unit (multiplier and adder) will make their effects appear at its output ports in terms of incorrect results which are different from the expected results. A testable design of the PE functional unit is required. A faulty flip flop is assumed to store erroneous values given combinations of inputs and initial states. A register consists of flip flops, and therefore, its fault model is the same as that of a flip flop. A faulty multiplexer, demultiplexer, selector, switch, or link is assumed to pass incorrect data. For each bit line of a control register scan path or data path, the test pattern $T = "00110"$ or $\bar{T} = "11001"$, which includes all possible input sequences (00), (01), (11), and (10), can be applied to detect faults in the flip flops and links.
Figure 4.7. Testing and diagnosis process.
4.1.5.2. Testing of Control Registers in PEs and Switches

For each horizontal PE track, there are two control register scan paths; one consists of \( C_d \)'s and the other of \( C_m \)'s. As for each vertical PE or link track, there is a 2-bit (bit-0 and bit-1) control register scan path consisting of \( SW_{reg} \)'s. By scanning the test pattern \( T \) or \( \overline{T} \) to each bit line of all control register scan paths, all control registers in PEs and switches can be tested for stuck-at faults by observing the output which should be the same as the input test pattern. For bridging faults, the test patterns \( T \) and \( \overline{T} \) are scanned into the even-numbered bit line and the odd-numbered bit line of a multi-bit control register scan path respectively to detect a short between two adjacent bit lines since \( T \) and \( \overline{T} \) are complements to each other. The control register testing procedure is as follows.

**Procedure Control_register_test**

1. Perform Steps 2 and 3 concurrently.

2. For every horizontal PE track \( i, 1 \leq i \leq M \), apply \( T \) and \( \overline{T} \) concurrently to the \( C_d \)-based and \( C_m \)-based scan paths respectively. If any test result is incorrect, the horizontal PE track \( i \) is labeled as faulty.

3. For every vertical PE or link track \( j, 1 \leq j \leq Q \), apply \( T \) and \( \overline{T} \) concurrently to the bit-0 line and bit-1 line of the \( SW_{reg} \)-based scan path respectively. If any test result is incorrect, the vertical PE or link track \( j \) is labeled as faulty.

Since the number of registers of a scan path is known, single and multiple flip flop faults can be detected by inspecting the values and the sequence of the test results. Once a control register scan path is determined to be faulty, the corresponding horizontal PE track, or vertical PE or link track needs to be switched out via the phase 1 reconfiguration process.
4.1.5.3. Switch and Link Testing

Switches and links are usually assumed fault-free in the literature for array testing. This assumption is not realistic since the switches and links can be faulty and cause false diagnosis. The main purpose of the switch and link testing is to find fault-free data paths such that test patterns and test results can be transmitted to and from each PE. Other methods of switch and link testing, which might be time consuming, were presented in [7, 46]. The host array is first converted to the lattice array. The test patterns $T$ and $\overline{T}$ are applied to every even-numbered bit line and odd-numbered bit line of a data path respectively. The five matrices, $RSV$, $CSV$, $RSM$, $CSM$, and $ASM$, are used to record test results. Initially, all the entries in $RSV$ and $CSV$ are assigned the value $x$, and all the entries in $ASM$ are assigned the value $0$. The entries in $RSM$ and $CSM$ are normally assigned the value $x$. But for an entry in $RSM$ and $CSM$ corresponding to a PE which has only the horizontal and vertical bypassing capability, or no PE or switch at that location, they are assigned the value $y$. The switch and link testing including three procedures is described as follows.

(1) Level-1 test

This procedure is to apply level-1 operators on level-1 data paths in the lattice array and record the test results in $RSV$ and $CSV$.

Procedure Level-1 test

1. For every $i$ and $j$, operating in parallel, $1 \leq i \leq M$, $1 \leq j \leq Q$, scan in a string of '0' to set every $C_d = '0'$, a string of '1' to set every $C_m = '1'$ of the horizontal PE track $i$, and a string of "00" to set every $SW_{reg} = "00"$ of the vertical PE or link track $j$.

2. For every $i$ and $j$, operating in parallel, $1 \leq i \leq P$, $1 \leq j \leq Q$, perform Step 3 and Step 4 concurrently.
3. Apply $RO_i$ on $RP_j$. If the test result is incorrect, then $RSV[i] := 1$ else $RSV[i] := 0$.

4. Apply $CO_j$ on $CP_j$. If the test result is incorrect, then $CSV[j] := 1$ else $CSV[j] := 0$.

(2) Level-2 test

This procedure is to apply level-2 operators on level-2 data paths in the lattice array and record the test results in $RSM$ and $CSM$.

Procedure Level-2_test

1. For every $i$ and $j$, $1 \leq i \leq P$, $1 \leq j \leq Q$, if the value of $RSM[i, j]$ is $x$, then set $SW_{reg} = \text{"01"}$ at $(i, j)$ and $SW_{reg} = \text{"00"}$ at the rest of switches in column $j$ of the logic array, and perform Steps 2 and 3 concurrently; otherwise, do nothing.

2. Apply $RO_{ij}$ on $RP_{ij}$. If the test result is incorrect, then $RSM[i, j] := 1$ else $RSM[i, j] := 0$.

3. Apply $CO_{j,i}$ on $CP_{j,i}$. If the test result is incorrect, then $CSM[i, j] := 1$ else $CSM[i, j] := 0$.

(3) ASM determination

Based on $RSV$, $CSV$, $RSM$, and $CSM$, ASM can be derived through pattern matching to identify any possible switch or link fault.

Procedure ASM_determination

1. Set $counter = 0$.

   For every $i$, $1 \leq i \leq P$, if $RSV[i] = 0$, then $counter := counter+1$, otherwise do nothing.

   For every $j$, $1 \leq j \leq Q$, if $CSV[j] = 0$, then $counter := counter+1$, otherwise do nothing.

   If $counter = P + Q$, then print "No faulty switch or link!" and stop.
2. Set flag = true;

For every \( i^* \) and \( j^* \), \( 1 \leq i^* \leq P, 1 \leq j^* \leq Q \), if the following conditions hold:

1. \( RSV[i^*] = 1 \) and \( CSV[j^*] = 1 \).
2. \( RSM[i, j] = 1 \) or \( y \) for \( i = i^*, j = j^* \) to \( Q \) and \( RSM[i, j] = 1 \) or \( y \) for \( i = 1 \) to \( i^*-1, j = j^* \),
3. \( CSM[i, j] = 1 \) or \( y \) for \( i = i^* \) to \( P, j = j^* \) and \( CSM[i, j] = 1 \) or \( y \) for \( i = i^*, j = 1 \) to \( j^*-1 \),

then \( ASM[i^*, j^*] := 1 \) and flag := false, else do nothing.

If flag = false, then print "Faulty switches detected!" and stop.

3. For every \( i^* \) and \( j^* \), \( 1 \leq i^* < P, 1 \leq j^* < Q \), if \( RSV[i^*] = 1 \) and one of the following conditions holds:

1. \( RSM[i, j] = 0 \) or \( y \) for \( i = 1 \) to \( i^* - 1, j = j^* \) to \( Q \), \( RSM[i^*, j^* + 1] = 1 \),
2. \( CSV[j^* + 1] \neq 1 \), and \( RSM[i, j] = 1 \) or \( y \) for \( i = i^*, j = j^* + 2 \) to \( Q \),
3. \( CSM[i, j] = 1 \) or \( y \) for \( i = 1 \) to \( i^* - 1, CSM[i^*, j^*] = 1 \),
4. \( CSV[j^*] \neq 1 \), and \( CSM[i, j] = 0 \) or \( y \) for \( i = 1 \) to \( i^*-1, j = j^* + 1 \) to \( Q \),

then \( ASM[i^*, j^*] := 3 \) and \( ASM[i^*, j^* + 1] := 3 \).

4. For every \( i^* \) and \( j^* \), \( 1 \leq i^* < P, 1 \leq j^* < Q \), if \( CSV[j^*] = 1 \) and one of the following conditions holds:

1. \( RSM[i, j] = 1 \) or \( y \) for \( i = 1 \) to \( i^*-1, j = j^* \), \( RSM[i^*, j^*] = 1 \),
After $ASM$ is obtained, it is then passed to the phase 2 reconfiguration process if there is any faulty switch or link. The reconfiguration process restructures the lattice array according to the determined fault locations. If the reconfiguration process succeeds, the switch and link testing process will be executed again to ensure no faulty switches and links in the lattice array. The number of iterative executions of these two processes, the switch and link testing process and the reconfiguration process, is expected to be small given the low probability of switch or link faults.

### 4.1.5.4. Diagnosis of Multiple PE-Faults

After all switch and link faults are switched out, the lattice array is converted to the logic array for the PE fault diagnosis. Assume the size of the logic array is $m \times n$. The PE fault diagnosis algorithm is based on the parallel partition approach. One MDB is tested at a time while the PEs in the same MDB are diagnosed in parallel. Multiple PE-faults in the logic array can be detected and located. The diagnosis process is described as follows. All the PEs in the logic array are first configured to operate in the bypass mode. The PEs in an MDB under test

\[
RSV[i^*] \neq 1, \text{ and } RSM[i, j] = 0 \text{ or } y \text{ for } i = i^*+1 \text{ to } P, j = j^*,
\]

(2) \[
RSM[i, j] = 1 \text{ or } y \text{ for } i = 1 \text{ to } i^*-1, j = j^*, RSM[i^*, j^*] = y, \text{ and}
\]

\[
RSM[i, j] = 0 \text{ or } y \text{ for } i = i^*+1 \text{ to } P, j = j^*,
\]

(3) \[
CSM[i, j] = 0 \text{ or } y \text{ for } i = 1 \text{ to } i^*, j = j^*, CSM[i^*+1, j^*] = 1,
\]

\[
RSV[i^*+1] \neq 1, \text{ and } CSM[i, j] = 1 \text{ or } y \text{ for } i = i^*+2 \text{ to } P, j = j^*,
\]

(4) \[
CSM[i, j] = 0 \text{ or } y \text{ for } i = 1 \text{ to } i^*, j = j^*, CSM[i^*+1, j^*] = y, \text{ and}
\]

\[
CSM[i, j] = 1 \text{ or } y \text{ for } i = i^*+2 \text{ to } P, j = j^*,
\]

then $ASM[i^*, j^*] := 3$ and $ASM[i^*+1, j^*] := 3$. 
are then configured to the normal mode. By applying the parallel partition algorithm, \( \min(m,n) \) PEs in an MDB are tested simultaneously. The diagnosis process for a PE in a systolic array consists of three parts: (1) control flip flop testing, (2) internal data path testing, and (3) PE functional unit testing. Note that the testings of control flip flops and internal data paths have been handled during the control register testing and the switch and link testing. The diagnosis process for \( PE_a \) is described below to demonstrate the PE functional unit testing which is divided into the multiplier test and the adder test.

(1) \( PE_a \) multiplier test

Procedure \textit{multiplier\_test}

1. Set \( C_d := 1 \) and \( C_m := 0 \).
2. Scan 0 into \( R_x \) via \( X_{in} \).
3. Set \( C_d := 0 \).
4. Scan the multiplier test patterns into \( W \) and \( R_y \) via \( X_{in} \) and \( Y_{in} \) respectively.
5. Allow sufficient time for the combinational logic to settle and check the output \( X_{out} \).

(2) \( PE_a \) adder test

Procedure \textit{adder\_test}

1. Set \( C_d := 0 \) and \( C_m := 0 \).
2. Scan 1 into \( W \) via \( X_{in} \).
3. Set \( C_d := 1 \).
4. Scan the adder test patterns into \( R_x \) and \( R_y \) via \( X_{in} \) and \( Y_{in} \) respectively.
5. Allow sufficient time for the combinational logic to settle and check the output $X_{out}$.

The testings of multiplier and adder apply the idea of path sensitization at the register level. The testing processes for $PE_b$ and $PE_c$ can be similarly treated. The algorithm for diagnosing all multiple PE-faults for the logic array is summarized in Figure 4.8. After all PE faults are located, the phase 3 reconfiguration process is applied to switch out faulty PEs and the target array is obtained if the reconfiguration succeeds.

4.1.5.5. Example

This example is to demonstrate how the switch and link testing and the parallel partition approach work. Figure 4.9(a) is a $PE_a$-based host array $<6, 6, 1, 1, 2, 0, 0>$ with faulty switches, links, and PEs. An "x" indicates that the corresponding switch, link, or PE function unit is faulty. An "xx" indicates that the corresponding PE has a fatal fault which makes its bypass (register) links and functional unit unusable. The $ASM$ in Figure 4.9(b) is the

**Algorithm Diagnosis_of_Multiple_PE_Faults**

Begin

1. Activate the BRLs of all the PEs in the $m \times n$ logic array;

2. $For$ MDB $b = 1$ to $max(m, n)$ $do$

   2.1. Deactivate the BRLs of PEs in MDB $b$;
   2.2. Test $min(m, n)$ PEs in MDB $b$ in parallel:
      2.2.1 Perform the multiplier test;
      2.2.2 Perform the adder test;
   2.3. If the test results of a PE[$i, j$] disagree with the expected results
      Then the PE[$i, j$] is marked as faulty in $ASM[i, j]$;
      $End-if$;
   2.4. Activate the BRLs of PEs in MDB $b$;

$End-for$.

$End$.

Figure 4.8. Multiple PE-faults diagnosis algorithm.
Figure 4.9. A host array and its diagnosis matrices.
representation of the faulty host array in Figure 4.9(a). The different types of faults are handled at different phases of diagnosis and reconfiguration processes. After applying the first iteration of level-1_test and level-2_test, $RSV$, $CSV$, $RSM$, and $CSM$ are as shown in Figure 4.9(c)-(f) respectively. The switch faults at $ASM[6, 11]$ and $ASM[9, 10]$, and the PE fault at $ASM[6, 10]$ are located during the first iteration of the switch and link testing. The link faults at the link between $ASM[10, 6]$ and $ASM[11, 6]$, and at the link between $ASM[11, 6]$ and $ASM[12, 6]$ are located during the second iteration of the switch and link testing. The rest of PE faults is located during the multiple PE-faults diagnosis. Note that the links at $(9, 6, 10, 6)$ and $(12, 6, 13, 6)$ are also determined to be faulty. This is because the $PE_a$ has the horizontal and vertical bypassing capability only. We cannot differentiate the faults at a PE’s horizontal (vertical) input link and output link. Nevertheless, this kind of false diagnosis will not affect the reconfigurability of the array. If a PE’s horizontal (vertical) input link is faulty, the PE’s horizontal (vertical) output link will not be available for reconfiguration anyway, and vice versa.

4.1.5.6. Discussion

The switch and link testing procedures involve the integration of the diagnosis process and the reconfiguration process to determine whether the switches and links in the lattice array are either fault-free or unreconfigurable. Switch faults are located before link faults. The reconfiguration process which includes spare allocation and actual reconfiguration has been developed in [2]. The divide and conquer strategy is applied to both the diagnosis process and the reconfiguration process to reduce diagnosis and reconfiguration complexities. The integration of the switch and link testing process and reconfiguration process is mandatory if switches and links can be faulty. That is, in order to find fault-free data paths to facilitate the diagnosis of multiple PE-faults, reconfiguration is necessary if data paths can be faulty. Although switch
and link faults are rare, they do exist. The proposed switch and link testing method can correctly recognize these faults. Since a switch or a link is much simpler and smaller than a PE, the probability of switch or link faults should be far less than that of PE faults. Therefore, only level-1 and level-2 operators are applied to determine any faulty switch or link. A study [47] showed the following yield statistics: PE 30-65%, switch 99%, and link (or wire) 95%. The fault diagnosis method used in the switch and link testing process is thus well justified. It is suitable for large arrays where the diagnosis of large number of switches and links is time consuming. The switch and link testing can diagnose all the functional faults due to physical stack-at-0, stack-at-1, and bridging faults. The fault coverage of the multiple PE-faults diagnosis algorithm is the same as that of a single PE test. The fault coverage of a single PE test is expected to be high because the PE is designed to be easily testable.

4.1.6. Extension to Other Parallel Architectures

The switch and link testing process can be applied to any multiple link track PE-switch lattice array. The parallel partition approach can also be applied to other parallel architectures. As long as MDBs exist in a parallel architecture and the PE structure meets the general PE model in Figure 4.2, all multiple PE-faults in the architecture can be diagnosed by the parallel partition approach. Some representative architectures are: square arrays, torus arrays, triangular arrays, hexagonal arrays, binary trees, m-ary trees, multistage interconnection networks such as Omega networks, and FFT networks [17,43,48,49]. Figure 4.10 shows five example architectures. The dotted lines in an architecture are its diagonal groups.

The data structures of the binary tree, m-ary tree, Omega network and FFT network can be linked lists. By using linked lists, it is easy to find out MDBs from the head to the tail of the
linked lists. For example, a linked list of the binary tree (Figure 4.10(d)) starts from the root PE to a leaf PE. By tracing all such linked lists from head to tail in parallel, the PEs at the same trace step are in the same MDB. The test patterns for the binary tree are applied through the input port of the root PE to all PEs in an MDB under test. Although these PEs share part of the input data paths, they have disjoint output data paths. As we can see, the MDBs for a tree structure correspond to the levels of the tree; hence, the number of MDBs is the height of the tree. The MDBs in an Omega network or FFT network correspond to the stages in the network.

Table 4.1 shows the summary of speed-ups of the parallel partition approach over the serial testing method for various parallel architectures. The serial testing refers to the testing of
Table 4.1. Speed-up summary for various parallel architectures.

<table>
<thead>
<tr>
<th>Parallel Architecture</th>
<th>Number of MDBs</th>
<th>Number of PEs</th>
<th>Speed Up</th>
<th>Application Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>square array (n x n)</td>
<td>n</td>
<td>$n^2$</td>
<td>$O(n)$</td>
<td>dynamic programming</td>
</tr>
<tr>
<td>rectangular array (m x n)</td>
<td>$\max(m,n)$</td>
<td>$mn$</td>
<td>$O(\min(m,n))$</td>
<td>matrix multiplication</td>
</tr>
<tr>
<td>torus array (n x n)</td>
<td>n</td>
<td>$n^2$</td>
<td>$O(n)$</td>
<td>transitive closure</td>
</tr>
<tr>
<td>triangular array (n x n / 2)</td>
<td>n</td>
<td>$n^2/2$</td>
<td>$O(n)$</td>
<td>inversion of triangular matrix</td>
</tr>
<tr>
<td>hexagonal array (n x n)</td>
<td>n</td>
<td>$n^2$</td>
<td>$O(n)$</td>
<td>L-U decomposition</td>
</tr>
<tr>
<td>binary tree ($</td>
<td>V</td>
<td>$)</td>
<td>$\lceil \log_2(</td>
<td>V</td>
</tr>
<tr>
<td>m-ary tree ($</td>
<td>V</td>
<td>$)</td>
<td>$\lceil \log_m(</td>
<td>V</td>
</tr>
<tr>
<td>Omega network (n-input)</td>
<td>$\log_2 n$</td>
<td>$(n/2)\log_2 n$</td>
<td>$O(n)$</td>
<td>interconnection network</td>
</tr>
<tr>
<td>FFT network (n-input)</td>
<td>$\log_m n$</td>
<td>$(n/m)\log_m n$</td>
<td>$O(n)$</td>
<td>signal processing</td>
</tr>
</tbody>
</table>

PEs one by one such as the wafer probe testing. The time complexity of the PE fault diagnosis algorithm for a parallel architecture is proportional to the number of MDBs. The minimum speed-up of the parallel partition approach over the serial testing for the parallel architectures in Table 4.1 is $O(|V|^{16})$, where $|V|$ is the size of an architecture. Parallel architectures with over 300 PEs per wafer can be fabricated with current technology [49]. This means that the parallel partition approach can speed up the diagnosis by 17.3 times with no or very little hardware
overhead. With the advancement of current technologies, significant increase in speed-up can be expected.

4.1.7. Summary

A systematic and efficient fault diagnosis approach for reconfigurable VLSI/WSI array architectures has been presented. The basic idea is to utilize the output data path independence among a subset of processing elements (PEs) based on the topology of the array under test. The "divide and conquer" technique is applied to reduce the complexity of test application and enhance the controllability and observability of an array. The array under test is divided into nonoverlapping maximal diagnosis blocks (MDBs). Those PEs in the same MDB can be diagnosed concurrently. The problem of finding MDBs is shown equivalent to a generalized Eight Queens problem. Three types of PEs, which are designed to be easily testable and reconfigurable, are used to show how to apply this approach. The main contribution originating from this approach is an efficient switch and link testing procedure, and a novel PE fault diagnosis approach which can speed up the testing by at least $O(|V|^{|V|/2})$ for the parallel architectures considered in this section, where $|V|$ is the number of PEs. The significance of our approach is its ability to detect as well as to locate multiple PE, switch, and link faults with little or no hardware overhead.
4.2. Self-Comparison Approach

4.2.1. Introduction

With the rapid progress of VLSI technologies, the chip sizes are to increase substantially, perhaps up to the level of WSI. With the increase in integration complexity of VLSI/WSI, the yield problem becomes very acute. A major method of increasing yield is to incorporate redundancy into a chip or whole wafer. But before faulty elements can be replaced by spare elements, faults have to be located. Once the faults are located, the next step is the fault replacement. The fault replacement stage consists of two steps: spare allocation and actual reconfiguration [2, 3].

It has been shown that the fault detection problem is NP-complete, which means there is no known polynomial time algorithms to solve it. The fault diagnosis problem is even harder. To overcome this problem, design for diagnosis [50] is one way of solutions. In this section, another novel and effective methodology for diagnosing faulty PEs in an array architecture is presented by including a small amount of extra hardware for diagnosability. Other parallel architectures [17, 43] are also amenable to this methodology. The self-comparison approach is the main skeleton of the diagnosis process. Although the comparison method was often used for adjacent PE comparison in the literature for system level diagnosis, it could only detect PE functional faults or locate a single PE fault [4, 5]. Cyr et al. [51] used an analog type of majority comparator to locate faulty PEs by voting among several PEs. The use of a majority comparator raises the concern of array locality. Kim and Reddy [52] designed an easily testable systolic array, but the array design was application specific. Choi et al. [53] proposed a token-triggered systolic diagnosis method for wafer scale arrays. In their design, extra hardware
comparable to the design of our approach is used, but its fault locatability depends on the number of PE comparisons. The self-comparison approach can diagnose multiple PE-faults in the array simultaneously and still maintains the locality of array operations. It is application independent and does not have the fault locatability problem.

The remaining subsections is organized in the following manner. The array model is described first. It includes the PE configuration and the selector connection models. Second, the fault model is presented which includes PE, switch, and link faults. Third, the design for diagnosability circuitry and the fault diagnosis algorithm are described. The area overhead and time complexity of the design are analyzed next followed with a summary.

4.2.2. Array Model

The two-dimensional array model shown in Figure 4.11, which is a special case of the general array model presented in Chapter 3 with \( S = 2 \) and \( T_h = T_v = 1 \), is used to illustrate the self-comparison approach. This approach is also applicable to the general array model. The squares represent PEs and the circles represent switches. The switches are interspersed with PEs to form a PE-switch lattice [54]. The switch design adopts the design in Chapter 3. The switches are used to facilitate array reconfiguration. Each PE is designed to be easily diagnosable as shown in Figure 4.12. The relationship of inputs and outputs in the PE is:

\[
X_{out} = f_x(X_{in}, Y_{in})
\]

\[
Y_{out} = f_y(X_{in}, Y_{in}).
\]

The functions, \( f_x \) and \( f_y \), of the functional block in the PE are application specific. Four selectors (marked as E, W, S, and N) and two comparators (\( C_x \) and \( C_y \)), two registers (\( R_x \) and \( R_y \)), one error flag flip flop (ER), and one OR gate are added to enhance diagnosability. Note that in
systolic arrays the PE may have a register in each data path. The ERs in a horizontal PE track can be connected as a shift register through the interconnections of $ER_{in}$ and $ER_{out}$. The error
information in the ERs can then be scanned out for the fault replacement process. The design for diagnosis criteria will be described later. The selector connection models are shown in Figure 4.13. There are two types of selectors: Type 1 and Type 2. The Type 1 selector is a demultiplexer and has two connection patterns (Types 1.1 and 1.2). The Type 2 selector is a multiplexer and has two connection patterns (Types 2.1 and 2.2). The W and N selectors are Type 1 selectors, while the E and S selectors are Type 2.

Each PE in the array has two operating modes: the normal mode and the test mode. In the normal mode, the selectors in the PE are controlled to direct the regular data flow. That is, the settings for E-selector, W-selector, S-selector, and N-selector are Types 2.2, 1.1, 2.2, and 1.1 respectively. In the test mode, the selectors in the PE are controlled to proceed to a two-phase testing. In Phase 1, the selector settings are controlled as E-selector (Type 2.1), W-selector (Type 1.1), S-selector (Type 2.1), and N-selector (Type 1.1). In Phase 2, the selector settings
are changed to E-selector (Type 2.1), W-selector (Type 1.2), S-selector (Type 2.1), N-selector (Type 1.2). The above operations are performed through the appropriate control of the selectors in the PE for the desired data flow.

4.2.3. Fault Model

The fault model includes faults in the PEs, switches and links. The detected switch and link faults are modeled as the adjacent PE faults. The selector faults will manifest as link faults. The other added circuits are simple, and hence can be assumed to be fault-free. If necessary, these circuits, such as comparators, can be designed as self-checking circuits.

4.2.3.1. PE Faults

PE functional faults are considered here. A PE is faulty if for a certain input it produces an output which is different from the expected output. In the diagnosis process, every PE in the array is applied the same test patterns concurrently. The size of test patterns depends on the complexity of the PE.

4.2.3.2. Switch and Link Faults

A faulty switch or link passes incorrect data. Switch faults and link faults are not distinguished in the diagnosis process. They are modeled as adjacent PE faults which will be detailed in the diagnosis process.
4.2.4. Design and Diagnosis

In this subsection, the design for diagnosis criteria and the fault diagnosis algorithm are described. The basis of the design for diagnosis and the diagnosis algorithm originates from the self-comparison approach. The main idea of this approach is designing an array architecture suitable for diagnosis independent of the array size.

4.2.4.1. Design for Diagnosis

In Figure 4.12, the four selectors (E, W, S, and N) are used to redirect data flow and create bypass links in the X and Y directions for each PE. By controlling selectors, the test patterns can be transmitted to all the PEs simultaneously through the bypass links. The selectors take the inputs to the PE as well as pass them to the two next PEs in the direction of data flow. The registers, \( R_x \) and \( R_y \), are used to temporarily hold the test results. The comparators, \( C_x \) and \( C_y \), are used to compare the test results with the expected test results. If there is a mismatch, the ER is set to "1" by \( C_x \) or \( C_y \) through the OR gate. The contents of the ERs can then be scanned out for the fault replacement process. The bypass links are used not only in the diagnosis process but also in the fault replacement process.

4.2.4.2. Fault Diagnosis

In order to diagnose the PEs in the array, switches and links need to be tested first. The switches and links are usually assumed fault-free in the literature for array testing. This assumption is not realistic since the switches or links might be faulty and can cause false diagnosis [46]. The main purpose of switch and link testing is to make sure that the test patterns can be transmitted to each PE through fault-free data paths. The switch and link testing is integrated with the PE diagnosis. A method of switch and link testing was presented in [46].
The testing was bit-oriented, which might be time consuming. It could only identify faulty regions.

Figure 4.14 shows the flowchart of WSI production process which includes the diagnosis process and the fault replacement process. This figure also shows how the design for diagnosis, the diagnosis process, and the fault replacement process interact with each other in the WSI production process. First the array is configured to operate in the test mode. In Phase 1 the test pattern, \((X, Y)\), is applied to every input port in the \(X\) and \(Y\) directions respectively. The test pattern then propagates to all PEs through the internal bypass links. Once the test pattern
arrives at a PE, the PE performs operations on it and produces the test result, \((X_o, Y_o)\). The test result is temporarily stored in the \(R_x\) and \(R_y\) registers. In Phase 2 the expected test result, \((X_o^*, Y_o^*)\), is applied to every input port in the \(X\) and \(Y\) directions respectively. It then propagates to each PE and is compared with the values in the \(R_x\) and \(R_y\) registers. If the inputs of \(C_x\) or \(C_y\) disagree, the corresponding comparator sets the ER to "1" to indicate there is an error. The details of fault diagnosis algorithm is shown in Figure 4.15.

After applying all the test patterns, the information in the ERs of PEs is scanned out to form an error matrix. The diagnosis process then checks the content of each row or column in the error matrix. Additional processing is needed if the rear of the content in a row or column has the pattern in (1), or a whole row or column has the pattern in (2):

1. \(0...0^"01"1...1\)
2. "1"1...1.

These two cases may result from the failures of switches or links. In the first case, two or more consecutive PEs at the rear of a row or column are indicated faulty. This may be caused by a faulty switch or link between the two PEs at the corresponding two quoted bit locations. As a result, all the subsequent PEs in the row or column receive incorrect input data. The diagnosis process replaces the pattern in (1) with \(0...0^"11"0...0\). That is, these two PEs are modeled as faulty. In the second case, all the PEs in a row or column are indicated faulty. It is possible that the switch or link at the left or top of the first PE at the corresponding quoted bit location is faulty. As a result, all the subsequent PEs receive incorrect input data. The pattern in (2) is replaced with "1"0...0. That is, the first PE is modeled as faulty. Figure 4.16(a) shows an example error matrix which is scanned out from the ERs in the array by the diagnosis algorithm. Row 2, Row 5, Column 4, and Column 7 have the above patterns. The diagnosis algorithm replaces these rows and columns with new contents according to the replacing rules. The
Procedure *Fault_Diagnosis*

Step 1. For every test pattern, perform Steps 2 to 4.

Step 2. `test_phase := "1"`.  
Apply the test pattern \((X_i, Y_i)\)  
to every input port simultaneously  
in the X and Y directions respectively. Each PE performs functional  
operations and generates \(X_o\) and \(Y_o\) simultaneously.

Step 3. For each PE do in parallel,  
\[
R_x := X_o  
R_y := Y_o. 
\]

Step 4. `test_phase := "2"`.  
Apply the expected test result \((X^*_o, Y^*_o)\)  
to every input port simultaneously  
in the X and Y directions respectively.  
For each PE do in parallel,  
If \(X^*_o = R_x\), then  
\[
C_x := 0 
\]
else  
\[
C_x := 1. 
\]
If \(Y^*_o = R_y\), then  
\[
C_y := 0 
\]
else  
\[
C_y := 1. 
\]
If \(C_x = 1\) or \(C_y = 1\), then  
\[
ER := 1 
\]
else  
\[
ER := 0. 
\]

Step 5. Scan out the content of each ER to form an error_matrix.  
If there is no "1" in the error_matrix, then go to Step 7  
else if the rear of the content in a row or column has the pattern in (1)  
or a whole row or column has the pattern in (2):  
(1) 0...0011...1  
(2) 11...1  
then  
replace (1) with 0...0110...0  
replace (2) with 10...0.

Step 6. Perform reconfiguration based on the error_matrix.  
Go to Step 1.

Step 7. Test succeeds.

Figure 4.15. Fault diagnosis algorithm.
modified error matrix is shown in Figure 4.16(b). The error matrix is then passed to the fault replacement process. The fault replacement process reconfigures the array according to the marked faulty PE locations. If the fault replacement process succeeds, the diagnosis process will be executed again to identify any more faulty PE in the array. The number of iterative executions of these two processes is expected to be small given the low probability of switch or link faults. This method integrates the diagnosis process and the fault replacement process to determine whether or not the array is reconfigurable. The integration of these two processes is mandatory if switches and links can be faulty. The fault replacement process has been developed in [2,3].

4.2.4.3. Discussion

If switches and links are assumed to be fault-free, the self-comparison approach can locate all the PE faults. Although switch and link faults are rare, they do exist. The heuristic method proposed above can handle these faults. Since a switch or a link is simpler and smaller compared to a PE, the probability of switch or link fault is far less than that of PE fault. One study [47] showed the following yield statistics: PE 30-65%, switch 99%, and link (or wire) 95%. The testing method for switches and links in our diagnosis process is thus well justified. It is
suitable for large arrays where the diagnosis of large number of switches and links is time consuming. Note that the design methodology based on the self-comparison approach can be extended for concurrent error detection and location in systolic arrays. We can design a systolic array such that it has an even number of horizontal (vertical) PE tracks. A systolic array usually has the odd (even) PEs being active while the even (odd) PEs being idle. We can utilize this property to proceed to two step comparisons. In Step 1, we make all the adjacent two odd-even PEs perform the same operations and then compare their results. In Step 2, we make all the adjacent two even-odd PEs perform the same operations and compare their results. Wrap around is assumed at the array boundaries to determine two adjacent PEs. That is, a boundary PE at one side of the array is assumed to be adjacent to the boundary PE at the other side [55]. By these two step operations we will be able to detect any fault, and locate PE faults depending on the fault locatability during normal systolic operations.

4.2.5. Analysis of Overhead and Complexity

The area overhead of the diagnosis circuitry and time complexity of the diagnosis algorithm are used to evaluate the proposed design methodology.

4.2.5.1. Area Overhead

The area overhead due to the extra circuits in each PE for fault diagnosis is not significant. The main area overhead of the design comes from the addition of two comparators and the associate logic for comparison. The four selectors and the associate control logic are not considered as overhead since they can be used to create bypass links for reconfiguration. Choi et al. [53] proposed a systolic array diagnosis method that also used bypass logic circuits and comparators
with similar overhead to the design of our approach. But our design has a simpler diagnosis algorithm and less time complexity. In addition, our design has much better diagnosis capability since it does not have the fault locatability problem that usually occurred in the testable designs based on the comparison method \([4, 5, 39, 53]\). The value of actual area overhead depends on the size of the PE functional block which can be either simple or complex in different applications.

4.2.5.2 Time Complexity of the Diagnosis Algorithm

For the convenience of analyzing the time complexity of the diagnosis algorithm, switches and links are assumed fault-free. The size of test patterns in the proposed diagnosis algorithm is the same as that of a single PE. Assume the size of the PE test patterns is \(N\). The number of the inputs to the array during diagnosis is \(2N\). Hence, the time complexity of the diagnosis algorithm is \(O(N)\). This means the testing time is independent of the array size. The testing time of the array is comparable to that of a single PE with additional time required to propagate the input data to each PE.

4.2.6. Summary

An efficient and application independent fault diagnosis approach in VLSI/WSI array architectures has been presented. Switch and link faults as well as PE faults are included in the fault model. Extra hardware is added to each processing element (PE) in the array to make the array easily diagnosable. By applying functional test patterns to each PE in the array, multiple PE-faults can be detected and located. In the proposed approach, all the PEs perform self-comparison operations simultaneously. The self-comparison approach is applicable to any
structure of PEs. The test pattern size is fixed, regardless of the array size. Although the whole design is for a unidirectional two-dimensional array, the same methodology can be extended to other kinds of parallel architectures. This approach is unique in the multiple-faults diagnosis capability as well as high fault coverage and small testing time. It is also amenable for concurrent error detection and location in systolic arrays.
CHAPTER 5.

OPTIMAL GROUP DIAGNOSIS PROCEDURES

5.1. Introduction

There are at least two motivations in conducting the research in this chapter. The first one is to minimize the diagnosis time and reduce the overall cost in the diagnosis and reconfiguration procedure for reconfigurable VLSI or WSI (wafer scale integration) architectures. It is stated in [56] that a problem is not to be considered solved in the mathematical sense until the structure of the optimal policy is understood. The concept of the mathematical solution is identical to the proper concept of a solution in the physical, economic, or engineering sense [56]. This is the second motivation for this research.

Optimal diagnosis procedures for coherent systems, such as $k$-out-of-$n$ structures, have been studied extensively [57,58,59,60,61,62,63]. A $k$-out-of-$n$ structure is said to be functional if at least $K_s = k$ out of $n$ elements are fault-free, and to have failed if at least $K_f = n-k+1$ out of $n$ elements are faulty. Although optimal diagnosis procedures for $k$-out-of-$n$ structures have been previously devised, these procedures test one element at a time (*single diagnosis*) and therefore, it may be time consuming for them to test a large system [57,59,61]. Speed-up in testing can be achieved if elements in a group which is a subset of the system can be tested in parallel. Optimal group testing procedures have thus been studied. A *group testing* procedure is a simultaneous test on a subset of elements with only two possible
outcomes. A good reading indicates that all the elements of the subset are non-defective, a bad reading shows that there is at least one defective element in the subset [64]. Most of the existing group testing procedures aim at locating one [65, 66, 67], two [68], or all defective elements [69, 70].

In this chapter, we investigate the optimal group diagnosis (OGD) policy for repairable VLSI/WSI array architectures with the objectives of tolerating manufacturing defects and enhancing yield. A group diagnosis procedure is a simultaneous diagnosis on a subset of elements such that each element in the subset can be identified as either fault-free or faulty. A sequence of decisions, which is called a multi-stage decision, has to be made in order to find an OGD sequence. An array is partitioned into disjoint maximal diagnosis blocks (MDBs) based on the parallel partition approach in Chapter 4 [26]. Due to the radial yield variations in semiconductor wafers [71], MDBs may have different yields from one another. Diagnosis cost can be a function of expected diagnosis time. An MDB with higher yield usually has higher diagnosis cost than an MDB with lower yield [57]. This is because an MDB with high yield has a high probability of having at least one fault-free element and must be tested by all the test patterns, while the testing of an MDB with low yield has a high probability that all the elements are faulty and is aborted when at least one fault is detected in every element. The goal of an OGD procedure is to find a sequential sequence of MDBs such that the accumulated diagnosis cost (ADC) in obtaining a possible reconfiguration solution is minimal. In a sequential sequence of MDBs, the next MDB to be tested depends on the current diagnosis outcome. This problem can be modeled as a \((t+1)\)-ary decision tree, where \(t\) is the size of an MDB. The complexity in dealing with a \((t+1)\)-ary decision tree is further reduced to that of handling a binary decision tree or a block-walking representation. Integrated diagnosis and reconfiguration processes are then investigated to fully utilize the OGD procedures.
This chapter is organized in the following manner. In Section 5.2, some terminologies are defined. The OGD procedures for reconfigurable array architectures and the integration of diagnosis and reconfiguration processes are presented in Section 5.3. In Section 5.4, the taxonomy and the complexity of the single diagnosis and the group diagnosis with various combinations of diagnosis strategies are analyzed. The simulation and comparison of the OGD procedures with the optimal single diagnosis (OSD) procedures are performed in Section 5.5 and followed with a summary.

5.2. Terminologies and Definitions

The array model presented in Chapter 3 is employed. Some related terminologies have been defined in Chapter 3. The degradation \((D)\) is defined as the ratio of the target array size over the logic array size. A binary decision tree \((DT_2)\) is a binary tree in which each of the internal nodes represents a binary decision and each of the external nodes represents a path decision outcome, either success \((s)\) or failure \((f)\). We use circles to represent internal nodes and squares to represent external nodes in a binary decision tree. We define that the root node in a binary decision tree is at level 0 and its children at level 1, and so on. A t-ary decision tree \((DT_t)\) can be similarly defined.

5.3. Optimal Group Diagnosis

We use array architectures to illustrate the OGD procedure which is based on the parallel partition approach. In Chapter 4, we have shown that the parallel partition approach can be extended to other parallel architectures. The OGD procedure can also be extended to other
parallel architectures. Based on the testing and diagnosis approach in Chapter 4 [26], a host array can be converted into a logic array by locating and then switching out the faulty switches and links. Now we are going to depict the OGD procedure to locate faulty PEs in a logic array. In an $M \times N$ logic array, the number of PEs in an MDB is $t = \min(M, N)$ and the number of MDBs is $n = \max(M, N)$ [26]. A logic array can be formalized as follows. Let a logic array be represented as $A_t = \{e_{ij} \mid i = 1, ..., n; j = 1, ..., t\}$, where $e_{ij}$ represents a PE. For each $e_{ij}$, $c_{ij}$ is its diagnosis cost and $p_{ij}$ is the priori probability that $e_{ij}$ is fault-free. Let $q_{ij} = 1 - p_{ij}$ and an MDB $b_i = \{e_{ij} \mid j = 1, ..., t\}$. Thus, $A_t = \{b_i \mid i = 1, ..., n\}$. For each MDB $b_i$, $c_i = \max(c_{ij} \mid j = 1, ..., t)$ is its diagnosis cost and $p_i$ is the average probability that all its PEs are fault-free; i.e., $p_i = \sum_{j=1}^{t} p_{ij} / t$. Also let $q_i = 1 - p_i$. Note that $c_i$ and $p_i$ are figures of merit to select an MDB to diagnose. Relabel the MDBs of the array such that

$$\frac{c_1}{p_1} \leq \frac{c_2}{p_2} \leq \cdots \leq \frac{c_n}{p_n} \quad (1)$$

and let $\pi$ be a permutation so that

$$\frac{c_{\pi(1)}}{q_{\pi(1)}} \leq \frac{c_{\pi(2)}}{q_{\pi(2)}} \leq \cdots \leq \frac{c_{\pi(n)}}{q_{\pi(n)}} \quad (2)$$

Also let $G_i = \{b_j \mid 1 \leq j \leq i\}$ and $B_i = \{b_{\pi(j)} \mid 1 \leq \pi(j) \leq i\}$. $G_i$ is the set of MDBs that should be diagnosed first if we want to locate $T_1$ fault-free PEs, where $(i-1)t < T_1 \leq i \cdot t$. $B_i$ is the set of MDBs that should be diagnosed first if we want to locate $T_2$ faulty PEs, where $(i-1)t < T_2 \leq i \cdot t$. Let $f(b_i)$ represent the number of fault-free PEs after the MDB $b_i$ has been tested. The value of $f(b_i)$ is in the range from 0 to $t$. 
5.3.1. Theoretical Basis

The OGD problem can be described formally as follows.

**Problem Specification:** For a logic array $A_t = <M, N, T_h, T_v, S, U, V>$, determine a sequential sequence $B$ of MDBs, with the terminating condition that either $T = UV$ out of $MN$ PEs are identified as fault-free or $MN - UV + 1$ out of $MN$ PEs are identified as faulty, and the ADC (accumulated diagnosis cost) is minimal.

We use $D_g(A_t, T)$ to represent a group diagnosis procedure for a logic array $A_t$. $|A_t|$ denotes the size of the logic array $A_t$. Note that $|A_t| = MN = nt$. $D_g(A_t, T)$ and $D_g(|A_t|, T)$ will be used interchangeably depending on applications. $D^*_g(A_t, T)$ represents an OGD procedure. We also use $D_g(|A_t|, T_1 - T_2)$ to represent a group diagnosis procedure which is amenable to the following diagnosis procedures: $D_g(|A_t|, T_1), D_g(|A_t|, T_1 + 1), ..., D_g(|A_t|, T_2)$, where $T_1 \leq T_2$. $D_s(n, k)$ represents a single diagnosis procedure for a $k$-out-of-$n$ structure, while $D^*_s(n, k)$ represents an OSD procedure. Figure 5.1 shows an example $3 \times 3$ logic array and its three MDBs, $b_1$, $b_2$, and $b_3$. Based on the radial yield variations in the wafer, we assume that $PE(2, 2)$ has the yield of 0.70 and each of the rest of the PEs has the yield of 0.60. Also, we assume that each PE has the diagnosis cost of 1.00. Table 5.1 shows the yield and diagnosis cost for each MDB. A group diagnosis procedure $D_g(A_t, T)$ can be modeled as a $(t+1)$-ary decision tree ($DT_{t+1}$). Figure 5.2 shows a diagnosis instance of the $DT_4$ for the logic array in Figure 5.1 with $t = 3$. The labels of an internal node in the tree represent the group diagnosis procedure and the corresponding single diagnosis procedure starting at that node, and the MDB $b_i$ to be tested. The number on a branch represents the number of fault-free PEs, $f(b_i)$. For example, at the root node of the $DT_4$ in Figure 5.2, $D_g(9, 6)$ represents a group diagnosis procedure that 6
Table 5.1. Yield and diagnosis cost for the example logic array.

<table>
<thead>
<tr>
<th>MDB</th>
<th>( b_1 )</th>
<th>( b_2 )</th>
<th>( b_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p_i )</td>
<td>0.60</td>
<td>0.60</td>
<td>0.63</td>
</tr>
<tr>
<td>( c_i )</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>( c_i / p_i )</td>
<td>1.67</td>
<td>1.67</td>
<td>1.59</td>
</tr>
<tr>
<td>( c_i / q_i )</td>
<td>2.50</td>
<td>2.50</td>
<td>2.70</td>
</tr>
</tbody>
</table>

out of 9 PEs need to be fault-free in order to get a status of *success* assuming that the size of the target array is 3 \( \times \) 2. This is equivalent to saying that 2 out of 3 MDBs need to be fault-free for a single diagnosis procedure \( D_s(3, 2) \). \( b_1 \) is the MDB to be tested. If all three PEs in \( b_1 \) are fault-free \( (f(b_1) = 3) \), the reduced procedure \( D_g(6, 3) \) (or \( D_s(2, 1) \)) is executed and \( b_3 \) is the next MDB to be diagnosed. A group diagnosis procedure is related to the corresponding single diagnosis procedure. We will first show their recursive definitions. Remember that a single diagnosis procedure is a test on an element with only two possible outcomes: fault-free and faulty. Hence, a \( D_g(n, k) \) can be represented by a \( DT_2 \) and described recursively in a general form by the following function:
Figure 5.2. A 4-ary decision tree.

\[ D_g(DT_2) = f(R_2, D_g(DT_{2}^l), D_g(DT_{2}^r)) \]  \hspace{1cm} (3)

where \( R_2 \) denotes a decision at the root node of the \( DT_2 \); \( D_g(DT_{2}^l) \) is the child procedure for the left binary decision subtree; and \( D_g(DT_{2}^r) \) is the child procedure for the right binary decision subtree. As for a \( D_g(A_t, T) \), it can be represented by a \( DT_{t+1} \) and described recursively in a general form by the following function:

\[ D_g(DT_{t+1}) = f(R_{t+1}, D_g(DT_{t+1}^{(b_i)})) \]  \hspace{1cm} (4)

where \( R_{t+1} \) denotes a decision at the root node of the \( DT_{t+1} \) and \( D_g(DT_{t+1}^{(b_i)}) \) represents \( t+1 \) possible child procedures (corresponding to the subtrees from the leftmost to the rightmost):

\( D_g(A_t - \{b_i\}, T-t), D_g(A_t - \{b_i\}, T-t+1), \ldots, \text{and} \ D_g(A_t - \{b_i\}, T). \)

Properties related to the group diagnosis procedure \( D_g(A_t, T) \) are derived as follows.
Lemma 5.1: A \((t+1)\)-ary decision tree for a group diagnosis procedure \(D_g(A_t, T)\) is equivalent to a binary decision tree for a single diagnosis procedure \(D_s(n, k)\), where \(n = \lceil A_t / t \rceil\) and \(k = \lceil T / t \rceil\).

Proof: From

\[
D_g(A_t, T) = f(R_{t+1}, D_g(A_t - t, T-t), D_g(A_t - t, T-t+1), \ldots, D_g(A_t - t, T)).
\]

we have

\[
D_s(\lceil A_t / t \rceil, \lceil T / t \rceil) = f(R_2, D_s(\lceil A_t / t - t \rceil, \lceil T-t / t \rceil), D_s(\lceil A_t / t - t \rceil, \lceil T-t+1 / t \rceil), \ldots,
\]

\[
D_s(\lceil A_t / t \rceil, \lceil T / t \rceil)).
\]

The leftmost subtree with \(k' = \lceil T-t / t \rceil\) and the rightmost subtree with \(k' = \lceil T / t \rceil\) have a difference of 1 in \(k'\). Therefore, the subtrees can be divided into two groups with \(k' = k-1\) and \(k' = k\) respectively. Based on the principle of optimality, those subtrees with the same parent node and the same \(D_s(n-1, k')\) should have the same testing sequence of MDBs and can thus be combined into a single subtree. Therefore,

\[
D_s(n, k) = f(R_2, D_s(n-1, k-1), D_s(n-1, k)) = D_s(n, k).
\]

For the sake of completeness, the following lemma in [57] with a slight modification is included here.

Lemma 5.2: The following single diagnosis procedure \(D_s(n, k)\) is optimal:

\[
D_s(n, k) = \begin{cases} 
(\text{Test}(b_1), D_s(n-1, k-1), D_s(n-1, k)) & \text{if } 0 < k \leq n \quad (a) \\
\text{success} & \text{if } k = 0 \quad (b) \\
\text{failure} & \text{if } k > n \quad (c)
\end{cases}
\]

where \(b_1\) is any MDB in \(G_k \cap B_{n-k+1}\).
This lemma says that $b_i$ is the MDB to be tested first and if $b_i$ is good, then the child procedure $D_g(n-1, k-1)$ is applied to test the remaining MDBs, else the child procedure $D_g(n-1, k)$ is applied to test the remaining MDBs. The diagnosis procedure terminates with the status of success if $K_s$ good MDBs have been found, or terminates with the status of failure if $K_f$ bad MDBs have been found. Remember that in a single diagnosis procedure a test on an MDB yields only two possible outcomes, good and bad. We then have the following theorem.

Theorem 5.1: The following group diagnosis procedure $D_g(A_1, T)$ is optimal:

$$D_g(A_1, T) = \begin{cases} 
(Diagnosis(b_i), D_g(A_1 - \{b_i\}, T - f(b_i)) & \text{if } 0 < T \leq |A_1| \\
\text{success} & \text{if } T = 0 \\
\text{failure} & \text{if } T > |A_1|
\end{cases}$$

where $b_i$ is any MDB in $G_k \cap B_{n-k+1}$.

Proof: This theorem follows immediately from Lemmas 5.1 and 5.2.

This theorem can be interpreted as follows. The MDB $b_i$ is diagnosed first. Depending on the number ($f(b_i)$) of fault-free PEs found in $b_i$, the child procedure $D_g(A_1 - \{b_i\}, T - f(b_i))$ is applied. The group diagnosis procedure terminates with the status of success if $T$ fault-free PEs have been found or terminates with the status of failure if $|A_1| - T + 1$ faulty PEs have been found. Note that Lemma 5.2 is a special case of Theorem 5.1 with $|b_i| = 1$, where $|b_i|$ is the size of $b_i$.

Corollary 5.1: For $T_1 \leq t$, $D^t_g(A_1, T_1) = (b_1, b_2, \ldots, b_n)$, which can be represented by a left degenerated $(t+1)$-ary decision tree, is equivalent to the diagnosis of an array which is functional if at least $T_1$ PEs are fault-free.
Proof: Since \( k = \lceil \frac{T_1}{t} \rceil = 1 \), \( G_k = G_1 \) and has only one element; i.e., \( G_1 \cap B_n = G_1 \). So, the sequence of MDBs to be tested is \( b_1, b_2, ..., b_n \).

Corollary 5.2: For \( |A_i| - t < T_2 \leq |A_i| \), \( D^*_g(A_i, T_2) = (b_{n(1)}, b_{n(2)}, ..., b_{n(n)}) \), which can be represented by a right degenerated \((t+1)\)-ary decision tree, is equivalent to the diagnosis of an array which is unfunctional if at least \( |A_i| - T_2 + 1 \) PEs are faulty.

Proof: Since \( k = \lceil \frac{T_2}{t} \rceil = |A_i| / t = n \), \( B_{n-k+1} = B_1 \) and has only one element; i.e., \( G_n \cap B_1 = B_1 \). So, the sequence of MDBs to be tested is \( b_{n(1)}, b_{n(2)}, ..., b_{n(n)} \).

The \( ADC(DT_{t+1}) \) of a group diagnosis procedure \( D_g(A_i, T) \) can be computed as follows:

\[
ADC(DT_{t+1}) = \sum_{i=1}^{n} \eta_i c_i
\]

where \( \eta_i \) is the probability of the appearance of the MDB \( b_i \) with the diagnosis cost \( c_i \) in the \((t+1)\)-ary decision tree \( DT_{t+1} \). Let \( \eta_{ij} \) denote the probability of having \( j \) fault-free PEs in \( b_i \), where \( 1 \leq i \leq n \) and \( 0 \leq j \leq t \). For instance, the \( ADC(DT_4) \) of the OGD procedure \( D_g^*(9, 6) \) in Figure 5.2 is:

\[
ADC(DT_4) = c_1 + (\eta_{12} + \eta_{11} + \eta_{10}) + \eta_{13}(\eta_{32} + \eta_{31} + \eta_{30})c_2 + (\eta_{13} + \eta_{12}(\eta_{23} + \eta_{22} + \eta_{21}) + \\
\eta_{11}(\eta_{12} + \eta_{11}) + \eta_{10}\eta_{123})c_3
\]

\[
= 2.767.
\]

This analytical value has been verified through simulation. In contrast to the OGD procedure, the \( ADC(DT_2) \) of the OSD procedure \( D_s^*(9, 6) \) via simulation is 7.424. The detailed simulation is described in Section 5.5.
5.3.2. Properties of Binary Decision Trees

Based on Lemma 5.1 and Theorem 5.1, the 4-ary decision tree in Figure 5.2 can be converted into a binary decision tree as shown in Figure 5.3. The following properties on the binary decision trees are useful for group diagnosis procedures as well as single diagnosis procedures. These properties have not been exploited in previous studies of OSD procedures.

Lemma 5.3: For a group diagnosis procedure \( D_g(A_t, T) \), the number \( N_s \) of success nodes \( s \) in the binary decision tree is \( N_s = C(n, k) \), where \( C(n, k) = \frac{n!}{(n-k)! k!} \).

\[
\text{Proof:} \quad \text{Based on Lemma 5.1, to lead to a success node, a diagnosis path in the binary decision tree must have } k \text{ left branches among } n \text{ branches. That is, the number of success nodes is } C(n, k).
\]

\( \square \)

Figure 5.3. A binary decision tree.
Lemma 5.4: For a group diagnosis procedure $D_g(A_1, T)$, the number ($N_f$) of failure nodes ($f$) in the binary decision tree is $N_f = C(n, n-k+1)$.

Proof: If the roles of fault-free PEs and faulty PE are exchanged, this lemma is equivalent to Lemma 5.3.

□

Theorem 5.2: For a group diagnosis procedure $D_g(A_1, T)$, the number of nodes generated in the binary decision tree is $2(C(n, k) + C(n, n-k+1)) - 1$.

Proof: From Lemmas 5.3 and 5.4, the number of external nodes in the binary decision tree is $C(n, k) + C(n, n-k+1)$. In a binary decision tree with $m$ internal nodes, there are always $m+1$ external nodes [42]. Thus, the number of internal nodes is $C(n, k) + C(n, n-k+1) - 1$. The theorem follows by the summation of the numbers of external nodes and internal nodes.

□

Corollary 5.3: For a group diagnosis procedure $D_g(A_1, T)$ with fixed $|A_1|$, the number of nodes in the binary decision tree is maximum when $k = n/2$ or $n/2+1$ if $n$ is even or $k = (n+1)/2$ if $n$ is odd, and is minimum when $k = 1$ or $n$.

Proof: This follows from that $C(n, k)$ is maximum when $k = (n-1)/2$ or $(n+1)/2$ if $n$ is odd or $k = n/2$ if $n$ is even [72] and $C(n, k)$ is minimum when $k = n$.

□

Corollary 5.4: For a group diagnosis procedure $D_g(A_1, T)$, the number of nodes generated in the binary decision tree is $O(n^{1/2})$ times less than that of a full binary decision tree which is generated when all MDBs are tested.
Proof: Without loss of generality, we assume that \( n \) is even. The maximum approximate value for \( C(n, k) \) is first derived. From Stirling's formula of approximation [72],
\[
n! = \sqrt{2\pi n} \left( \frac{n}{e} \right)^n,
\]
we have
\[
C(n, k) = \frac{n!}{k!(n-k)!} = \left( \frac{\sqrt{2\pi n}}{\pi n} \right) \left( \frac{n}{2e} \right)^n 2^n = O \left( \frac{2^n}{\sqrt{n}} \right), \text{ where } k = \frac{n}{2}.
\]
From Theorem 5.2, the number of nodes in the binary decision tree is \( O \left( \frac{2^n}{\sqrt{n}} \right) \). The number of nodes in a full binary decision tree is \( 2^n - 1 = O(2^n) \). Thus, the corollary is proved.

Theorem 5.3: Two group diagnosis procedures, \( D_g(A_{l1}, T_1) \) with \( t = t_1 \) and \( D_g(A_{l2}, T_2) \) with \( t = t_2 \), have the same binary decision tree structure if \( |A_{l1}| / t_1 = |A_{l2}| / t_2 \) and \( \lceil T_1 / t_1 \rceil = \lceil T_2 / t_2 \rceil \).

Proof: From \( n_1 = |A_{l1}| / t_1 = |A_{l2}| / t_2 = n_2 \) and \( k_1 = \lceil T_1 / t_1 \rceil = \lceil T_2 / t_2 \rceil = k_2 \), we have, \( D_s(n_1, k_1) = D_s(n_2, k_2) \). So, these two group diagnosis procedures have the same binary decision tree structure.

Theorem 5.4: Two group diagnosis procedures, \( D_g(A_{l1}, T_1) \) with \( t = t_1 \) and \( D_g(A_{l2}, T_2) \) with \( t = t_2 \), have the same number of nodes in their corresponding binary decision trees if \( |A_{l1}| / t_1 = |A_{l2}| / t_2 \) and \( \lceil T_1 / t_1 \rceil = \lceil A_{l2} / t_2 - \lceil T_2 / t_2 \rceil + 1 \).
Proof: From \( n_1 = |A_{I1}| \cdot \frac{t_1}{t_1} = |A_{I2}| \cdot \frac{t_2}{t_1} = n_2 \) and \( k_1 = \left\lceil T_1 / t_1 \right\rceil = |A_{I2}| \cdot \frac{t_2 - \left\lceil T_2 / t_2 \right\rceil + 1 = n_2 - k_2 + 1 \), these two group diagnosis procedures are equivalent if the roles of fault-free PEs and faulty PEs in one of the group diagnosis procedures are exchanged. So, these two procedures have the same number of nodes in their binary decision trees.

Lemma 5.5: For a group diagnosis procedure \( D_g(A_I, T) \), the number of success nodes in level \( i, k \leq i \leq n \), of the binary decision tree is \( C(i, k) - C(i-1, k) \).

Proof: For the binary decision tree, the total number of success nodes up to level \( i \) is \( C(i, k) \), and the total number of success nodes up to level \( i-1 \) is \( C(i-1, k) \), where \( k \leq i \leq n \). Thus, the number of success nodes in level \( i \) is \( C(i, k) - C(i-1, k) \).

Lemma 5.6: For a group diagnosis procedure \( D_g(A_I, T) \), the number of failure nodes in level \( i, n-k+1 \leq i \leq n \), of the binary decision tree is \( C(i, n-k+1) - C(i-1, n-k+1) \).

Proof: If the roles of fault-free PEs and faulty PEs are exchanged, Lemma 5.6 is equivalent to Lemma 5.5.

If the occurrences of external nodes in a binary decision tree are assumed equally probable and the branch length is assumed to be 1.00, then the average external path length leading to a success (failure) node can be computed. This information is useful to estimate the average number of MDBs being tested and the cost saving of the OGD procedure in comparison with the testing of all MDBs. Based on the results from Lemmas 5.5 and 5.6, we have the following theorem.
Theorem 5.5: For a group diagnosis procedure $D_g(A_t, T)$, the average external path length ($E_s$) leading to a success node in the binary decision tree is $E_s = \frac{nC(n, k) - \sum_{i=k}^{n-1} C(i, k)}{C(n, k)}$, and the average external path length ($E_f$) leading to a failure node in the binary decision tree is $E_f = \frac{nC(n, n-k+1) - \sum_{i=n-k+1}^{n-1} C(i, k)}{C(n, n-k+1)}$.

Proof: The total external path length leading to the success nodes in the binary decision tree is

$$kC(k, k) + (k+1)(C(k+1, k) - C(k, k)) + \cdots + n(C(n, k) - C(n-1, k))$$

$$= -C(k, k) - C(k+1, k) - \cdots - C(n-1, k) + nC(n, k)$$

$$= nC(n, k) - \sum_{i=k}^{n-1} C(i, k).$$

The value of $E_s$ is obtained by dividing the above value by the number of success nodes which is $C(n, k)$. Similarly, we can derive $E_f$ which is just a dual of $E_s$.

Corollary 5.5: For a group diagnosis procedure $D_g(A_t, T)$, the average external path length ($E$) of the binary decision tree is $E = \frac{(C(n, k)E_s + C(n, n-k+1)E_f)}{(C(n, k) + C(n, n-k+1))}$.

Proof: This can be derived directly from Theorem 5.5.

Corollary 5.6: For a group diagnosis procedure $D_g(A_t, T)$, the binary decision tree is:

\[
\begin{align*}
\text{symmetric} & \quad \text{if } k = n-k+1 & (a) \\
\text{left skew} & \quad \text{if } k > n-k+1 & (b) \\
\text{right skew} & \quad \text{if } k < n-k+1 & (c)
\end{align*}
\]
Proof: We know that $K_s = k$ and $K_f = n-k+1$. When $K_s = K_f$, the binary decision tree is symmetric because if the roles of fault-free PEs and faulty PEs are exchanged, the new binary decision tree should still have the same structure as the old one. When $K_s > K_f$, the binary decision tree is expanded toward the left side more than toward the right side. Case (c) is a dual of Case (b).

Based on the above discussion, the binary decision tree in Figure 5.3 generates $C(3, 2) = 3$ success nodes, $C(3, 2) = 3$ failure nodes, and $2(C(3, 2) + C(3, 2)) - 1 = 11$ nodes in total. The average external path length leading to a success node is $(3C(3, 2) - \sum_{i=2}^{2} C(i, 2)) / C(3, 2) = 2.667$. Similarly, the average external path length leading to a failure node is 2.667. Thus, the average external path length is 2.667, which is a good estimation of $ADC = 2.767$ as computed in Section 5.3.1.

5.3.3. Block-Walking Representation

In the following, transformation is made to reduce the exponential complexity of a binary decision tree representation to the polynomial complexity of a block-walking representation [57, 73]. The block-walking representation has been used in applied combinatorics to illustrate the number of shortest paths in which a walker can walk from one point to another point on a rectilinear grid [57, 73]. From Lemma 5.1 and Theorem 5.1, we know that an OGD procedure can be represented by a binary decision tree. Therefore, based on the results in [57], if the whole decision tree needs to be generated first, an OGD procedure $D_8^*(A_i, T)$ can be represented by a blocking-walking representation.
Theorem 5.6: For an OGD procedure $D^*_g(A_l, T)$, the number ($N_{bw}$) of nodes generated by the block-walking representation is $N_{bw} = n(k+1) - k(k-1) + 1$.

Proof: The number of nodes expanded toward the left edge is $K_s + 1$ and the number of nodes expanded toward the right edge is $K_f + 1$, so the number of nodes generated by the block-walking representation is $(K_s + 1)(K_f + 1) - 1 = n(k+1) - k(k-1) + 1$. \qed

For example, the binary decision tree for $D^*_g(9, 6)$ (or $D^*_g(3, 2)$) in Figure 5.3 can be represented by a block-walking representation as shown in Figure 5.4. Here, "n, k" is used to represent $D^*_g(n, k)$. The number of nodes in this block-walking representation is 8.

5.3.4. Integrated Diagnosis and Reconfiguration

Traditionally, the reconfiguration process is executed after the diagnosis process has identified all faulty elements. In order to take advantage of the OGD procedure, the integration of diagnosis and reconfiguration processes is necessary so that the diagnosis process and the reconfiguration process can interact with each other efficiently. The diagnosis process and

Figure 5.4. A block-walking representation.
reconfiguration process have to fully cooperate to get a reconfigurable solution or determine that a system is not reconfigurable as early as possible. A logic array is first passed to the diagnosis process. The diagnosis process diagnoses the logic array based on the OGD procedure. If the final outcome of the multi-stage decisions is "success" (reconfigurable), the error matrix for the logic array is passed to the reconfiguration process. The error matrix is an $M \times N$ matrix where each entry represents the status of a PE with a value of "1" (fault-free), "0" (faulty), or "u" (unknown). The reconfiguration process then attempts to repair the logic array. If the reconfiguration process succeeds, the target array is generated, otherwise it passes the reconfiguration results back to the diagnosis process. The reconfiguration results may contain information on directing the diagnosis process to diagnose the rest of the MDBs or only the selected MDBs. That is, after the first iteration of the diagnosis process, the reconfiguration process guides the diagnosis process to determine the next MDBs to be tested. The integrated diagnosis and reconfiguration procedure is summarized in Figure 5.5.

5.4. Taxonomy and Complexity Analysis of Diagnosis Strategies

The diagnosis strategies can be classified into single diagnosis and group diagnosis. Each type of diagnosis strategy can be further classified into random diagnosis, serial diagnosis, parallel diagnosis, and optimal diagnosis. The random diagnosis simply selects a unit arbitrarily to diagnose. A unit can be a PE or an MDB depending on whether it is the single diagnosis or the group diagnosis. The serial diagnosis follows the unit sequence in $B_n$, while the parallel diagnosis follows the unit sequence in $G_n$, to diagnose a unit. Depending on whether the whole decision tree is generated or not, the optimal diagnosis can be either static or dynamic. The static optimal diagnosis generates the whole decision tree first before proceeding to further diag-
Diagnosis process: based on optimal group diagnosis procedures

Figure 5.5. Integrated diagnosis and reconfiguration procedure.

nosis. For the static optimal diagnosis, a block-walking representation can be used in place of a binary decision tree. The dynamic optimal diagnosis generates the root node of the decision tree first, then depending the diagnosis outcome, it generates only one child node, and so on. Figure 5.6 show such classification of diagnosis strategies. The advantage of non-optimal diagnosis strategies is that there is no need to generate decision trees or block-walking representations and determine which unit to test; however their ADCs are higher than those of the optimal diagnosis strategies. The comparison among the random single diagnosis, serial single diagnosis, parallel single diagnosis, and static OSD can be found in [57].
We will concentrate on the analysis of the following four diagnosis strategies: static OSD, dynamic OSD, static OGD, and dynamic OGD. For the static OSD, it was shown in [57] that \( O(|V|^2) \) time is needed in constructing a block-walking representation with the assumption that the sorting can be done in \( O(|V|) \) by a radix sort, where \(|V|\) is the number of PEs. But in general, the sorting requires \( O(|V| \log |V|) \). Therefore, we need \( O(|V|^2 \log |V|) \) time to construct a block-walking representation for most cases. The space complexity of a block-walking representation for the static OSD procedure is \( O(nk) \) based on Theorem 5.6. But in general, \( k \) is of \( O(n) \). So the space complexity for the static OSD procedure is \( O(n^2) \) or \( O(|V|^2) \). For the dynamic OSD, the time and space complexities were shown to be \( O(|V|^3) \) and \( O(|V|) \) respectively in [60]. By using the parallel partition approach, the group diagnosis can speed up the testing by \( O(|V|^{1/2}) \) over the single diagnosis [26]. Hence, the time and space complexities of the static OGD and the dynamic OGD can be obtained by replacing \(|V|\) with \(|V|^{1/2}\) in those of the static OSD and the dynamic OSD respectively. The overall improvement of OGD over OSD is \( O(|V|^{1/2}) \).
5.5. Experiments: OGD versus OSD

The OGD and OSD procedures are implemented and simulated to evaluate their ADCs. The diagnosis cost is expressed in terms of the diagnosis time. The ADC does not include the time to construct the block-walking representation or to execute the optimal diagnosis procedure. That is, the ADC for the static optimal diagnosis procedure is the same as that of the dynamic optimal diagnosis procedure.

5.5.1. Simulation Set-up

Logic arrays with the radial yield variations are simulated. We use the manufacturing yield data from [71] to generate arrays with the radial yield variations. The yield data is the all-good yield at each chip position, relative to the wafer-averaged yield \( Y_{av} \) for FET (Field Effect Transistor) array product D on a 100-mm wafer in a newly qualified line [71]. \( Y_{av} \) is the average yield of wafers. Each chip position corresponds to a PE in our simulation. According to these data and various \( Y_{av} \) values, the simulation program, which is implemented in C running on SUN 3/260, generates different fault patterns for logic arrays with the radial yield variations. Each of the following simulation results is the average of ADCs on 10,000 simulation runs.

5.5.2. Experimental Results

We first simulate the cases that the diagnosis cost of each PE is uniform and is 1.00. Figure 5.7 shows the ADC comparison between OGD and OSD for a 10 \( \times \) 10 logic array with \( Y_{av} = 0.50 \). The target array size varies from 1 \( \times \) 1 to 10 \( \times \) 10. The cost of OSD is significantly
Figure 5.7. Comparison of ADCs between OGD and OSD ($Y_{av} = 0.50$).

bigger than that of OGD. The maximal diagnosis cost (MDC) occurs when the target array is
of size $7 \times 8$ for both OGD ($MDC = 9.686$) and OSD ($MDC = 92.236$). Note that the target
array with $MDC$ occurs at the degradation of $D = 0.56$ which is the smallest value that is higher
than the value of $Y_{av} = 0.50$. Table 5.2 summarizes the ADCs of OGD and OSD for the $10 \times$
$10$ logic arrays with $Y_{av} = 0.30$ and $0.65$. The selection of $Y_{av}$ values is based on the yield
statistics in [47]. For both OGD and OSD, the $MDC$s occur when the target array sizes are $6 \times$
$6$ and $8 \times 9$ for $Y_{av} = 0.30$ and $0.65$ respectively. Again, the target array with $MDC$ occurs with
$D = 0.36$ and $0.72$ which are the smallest values that are higher than the values of $Y_{av} = 0.30$
and $0.65$ respectively. So the occurrence of the target array with $MDC$ can be predicted by
knowing the value of $Y_{av}$. From Figure 5.7 and Table 5.2, we know that the $MDC$ occurs at a
larger target array for higher $Y_{av}$. This is because a larger size of target array can be obtained
when $Y_{av}$ is higher. Figure 5.8 shows the ADCs versus $Y_{av}$ for OGD and OSD in obtaining a $9$
$\times 9$ target array from a $10 \times 10$ logic array. The $MDC$s are incurred when $Y_{av} = 0.80$ for both
OGD ($MDC = 9.484$) and OSD ($MDC = 90.635$). This result with $D = 0.81$ is consistent with
the previous results. Note that the ADC of OSD increases more rapidly than that of OGD if $Y_{av}$
Table 5.2. ADC comparison with $Y_{av} = 0.30, 0.65$

<table>
<thead>
<tr>
<th>target array</th>
<th>$Y_{av}$</th>
<th>OGD</th>
<th>OSD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 x 1</td>
<td>0.30</td>
<td>1.014</td>
<td>2.489</td>
</tr>
<tr>
<td></td>
<td>0.65</td>
<td>1.000</td>
<td>1.139</td>
</tr>
<tr>
<td>5 x 6</td>
<td>0.30</td>
<td>9.072</td>
<td>86.566</td>
</tr>
<tr>
<td></td>
<td>0.65</td>
<td>4.499</td>
<td>39.936</td>
</tr>
<tr>
<td>6 x 6</td>
<td>0.30</td>
<td>9.709</td>
<td>92.332</td>
</tr>
<tr>
<td></td>
<td>0.65</td>
<td>5.319</td>
<td>48.360</td>
</tr>
<tr>
<td>6 x 7</td>
<td>0.30</td>
<td>9.290</td>
<td>87.704</td>
</tr>
<tr>
<td></td>
<td>0.65</td>
<td>6.149</td>
<td>56.970</td>
</tr>
<tr>
<td>8 x 8</td>
<td>0.30</td>
<td>6.062</td>
<td>53.673</td>
</tr>
<tr>
<td></td>
<td>0.65</td>
<td>9.196</td>
<td>88.227</td>
</tr>
<tr>
<td>8 x 9</td>
<td>0.30</td>
<td>4.790</td>
<td>41.918</td>
</tr>
<tr>
<td></td>
<td>0.65</td>
<td>9.714</td>
<td>92.226</td>
</tr>
<tr>
<td>9 x 9</td>
<td>0.30</td>
<td>3.410</td>
<td>28.554</td>
</tr>
<tr>
<td></td>
<td>0.65</td>
<td>7.696</td>
<td>65.964</td>
</tr>
<tr>
<td>9 x 10</td>
<td>0.30</td>
<td>2.082</td>
<td>15.316</td>
</tr>
<tr>
<td></td>
<td>0.65</td>
<td>4.336</td>
<td>32.029</td>
</tr>
<tr>
<td>10 x 10</td>
<td>0.30</td>
<td>1.000</td>
<td>1.235</td>
</tr>
<tr>
<td></td>
<td>0.65</td>
<td>1.000</td>
<td>1.000</td>
</tr>
</tbody>
</table>

Increases. From the simulation results, in general, the MDCs between OSD and OGD have a ratio in the order of 10 which is related the analytical results in the previous section. Also, in most cases, the MDCs for OGD and OSD decrease if $Y_{av}$ increases, because it would take less time to get a target array with MDC under higher $Y_{av}$.

We also simulate the situation of the non-uniform diagnosis cost for each PE depending on the PE status, either fault-free or faulty. Assume that the diagnosis cost of a fault-free (faulty) PE is $C_1$ ($C_0$). To facilitate the discussion, the following terminologies are defined. Since the diagnosis cost of each PE depends on the PE status, we need to have the cutoff yield definition. The cutoff yield ($Y_c$) is the yield that the diagnosis cost of a PE $e_{ij}$ is $C_1$ if its $p_{ij} > Y_c$ and is $C_0$, otherwise. The predicted accumulated diagnosis cost ($ADC_p$) is the ADC that is
based on the predicted diagnosis cost of each PE. The accumulated diagnosis cost deviation is defined as \( ADC_d = \frac{|ADC - ADC_p|}{ADC} \times 100\% \). The full accumulated diagnosis cost \( (ADC_f) \) is the ADC on which all the PEs (or MDBs) are tested. The cost saving of the ADC of OGD \( (ADC_g) \) over that of OSD \( (ADC_s) \) is \( ADC_{cs} = \frac{ADC_s}{ADC_g} \). After simulating different \( Y_c \) values, we find that the \( ADC_d \)'s for OGD and OSD are the smallest in overall when \( Y_c = Y_{av} \). So we will use the value of \( Y_{av} \) as the cutoff yield. All the following statistics for OGD and OSD are the simulation results in obtaining a 9 \( \times \) 9 target array from a 10 \( \times \) 10 logic array. Table 5.3 shows the \( ADC \) related statistics for OGD and OSD under the conditions of \( Y_{av} = Y_c = 0.30, 0.50, 0.65, C_1 = 1.00, \) and \( C_0 = 0.5, 0.25 \). The values of \( ADC, ADC_p, ADC_f, \) and \( ADC_{cs} \) increase if \( Y_{av} \) increases, while \( ADC_d \) decreases if \( Y_{av} \) increases. More PEs (or MDBs) will need to be tested to determine a feasible reconfiguration solution instead of getting early abort under high \( Y_{av} \). The OGD has a much smaller \( ADC_d \) than the OSD. Thus, the \( ADC \) for OGD is closer to the real optimal value than that of OSD in the sense that the predicted diagnosis cost of each MDB in OGD is more accurate than that of each PE in OSD. The \( ADC_{cs} \) decreases if \( C_0 \)
Table 5.3. ADC related statistics with $Y_{av} = 0.30, 0.50, 0.65$.

<table>
<thead>
<tr>
<th>$Y_{av}$</th>
<th>OGD ($C_0 = 0.50$)</th>
<th>OSD ($C_0 = 0.50$)</th>
<th>OGD ($C_0 = 0.25$)</th>
<th>OSD ($C_0 = 0.25$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ADCP$</td>
<td>3.410</td>
<td>24.397</td>
<td>3.410</td>
<td>21.897</td>
</tr>
<tr>
<td>$ADC_d$</td>
<td>0.009</td>
<td>0.257</td>
<td>0.013</td>
<td>0.520</td>
</tr>
<tr>
<td>$ADC_f$</td>
<td>9.920</td>
<td>66.918</td>
<td>9.880</td>
<td>50.377</td>
</tr>
<tr>
<td>$ADCC_{sr}$</td>
<td>5.741</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$Y_{av} = 0.50$

| $ADC$    | 4.930               | 32.944              | 4.930               | 27.954              |
| $ADCP$   | 4.931               | 37.922              | 4.931               | 35.422              |
| $ADC_d$  | 0.000               | 0.151               | 0.000               | 0.267               |
| $ADC_f$  | 9.999               | 78.194              | 9.998               | 67.291              |
| $ADCC_{sr}$ | 6.682           |                     |                     |                     |

$Y_{av} = 0.65$

| $ADC$    | 7.696               | 53.210              | 7.696               | 48.237              |
| $ADCP$   | 7.696               | 58.158              | 7.696               | 55.658              |
| $ADC_d$  | 0.000               | 0.093               | 0.000               | 0.154               |
| $ADC_f$  | 10.000              | 86.655              | 10.000              | 79.982              |
| $ADCC_{sr}$ | 6.914           |                     |                     |                     |

decreases. This is because the ADC of OSD is affected by the value of $C_0$ more than that of OGD. All the above results relate to the fact that in OGD the testing of an MDB can be aborted only when at least one fault is found in every PE of the MDB. One obvious result is that the ADCs for the cases of non-uniform PE diagnosis cost are less than those for the cases of uniform PE diagnosis cost. The simulation results also show that all the $MDC$s occur at $Y_{av} = 0.80$ for both OGD and OSD under various $C_0$'s. This fact with $D = 0.81$ is still consistent with the previous analysis, and is independent of the value of $C_0$. Figure 5.9 shows the relationship between $ADCC_{sr}$ and $Y_{av}$ with the variance of $C_0$ ($C_1 = 1$). In general, $ADCC_{sr}$ increases if $Y_{av}$ increases, although not smoothly, while $ADCC_{sr}$ decreases if $C_0$ decreases. That is, although the ADC difference between OGD and OSD becomes smaller if $C_0$ decreases, OGD
Figure 5.9. $ADC_{cs}$ versus $Y_{av}$.

still better than OSD in terms of $ADC$, especially when $Y_{av}$ is high. In summary, all the simulation results support the effectiveness of the OGD procedures.

5.6. Summary

We have addressed the problem of partitioning VLSI/WSI array architectures into disjoint maximal diagnosis blocks (MDBs) and finding an optimal group diagnosis policy for testing and locating faulty elements (modules) in these MDBs. The optimization criterion is to minimize the accumulated diagnosis cost in deriving a feasible reconfiguration solution. The method of partitioning an array is based on the parallel partition approach. The problem of finding an optimal group diagnosis procedure is modeled as a $(t+1)$-ary decision tree, where $t$ is the size of an MDB. Instead of dealing with a $(t+1)$-ary decision tree directly, the problem is further reduced to that of handling a binary decision tree or a block-walking representation. Properties related to the group diagnosis procedures and binary decision trees have been derived. We also present the strategies of integrating diagnosis and reconfiguration processes to take advantage of
the optimal group diagnosis procedures. The taxonomy and the complexity analysis of various
diagnosis strategies are investigated to illustrate the benefits of the optimal group diagnosis pro-
cedures. The analysis shows that the optimal group diagnosis procedure has an $O(|\mathcal{V}|^{1/2})$
improvement in time and space complexities over the optimal single diagnosis procedure, where
$|\mathcal{V}|$ is the number of processing elements (PEs) in an array. Simulation results are provided to
further support the effectiveness of our novel approach over other approaches. The optimal
group diagnosis procedures can be easily extended to other parallel architectures that are amen-
able to the parallel partition approach.
CHAPTER 6.

SIMULATION AND EVALUATION

6.1. Introduction

Only host arrays with PE faults are considered, which is usually assumed in the literature [8, 9], in the simulation and evaluation processes. The effect of faulty switches and links has been addressed in Chapter 4 [26], where the faulty switches and links can be identified. During reconfiguration, each faulty switch or link is lumped into one of the adjacent PEs and that PE is considered as faulty. One study [47] has the following yield statistics: 30-65% for PEs, 99% for switches, and 95% for links (or wires). Since the yields of switches and links are high, it is appropriate to use host arrays with only PE faults to illustrate the simulation and evaluation processes.

6.2. Reconfiguration Process

The reconfiguration process in Figure 6.1 includes three stages [74]:

(1) the execution of the reconfiguration program,
(2) the execution of the switching mechanism transformation program, and
(3) the generation of the VHDL target array.

The reconfiguration algorithms described in [25] are used to generate a logic matrix based on the error matrix for a host array. The transformation program takes the logic matrix as the input.
and transforms it into a characteristic matrix according to the selected switching mechanism. The characteristic matrix which contains switch settings and PE bypassing information can then be used as the control input to reconfigure the VHDL-based host array into the target array. The correctness of the reconfiguration process is verified through VHDL simulation. The transformation program consists of two parts: the transformation algorithm and the text to binary conversion program. The following discussion is based on a type-2 single link track PE-switch lattice model. For a PE's external connections, the following issues need to be addressed in the transformation algorithm: finding out its east and south neighbors (if available), and determining whether or not it is a boundary PE in the target array. The patterns of reconfiguration can be classified into three groups. They are row mapping, column mapping, and boundary mapping groups. Each group can be further divided into three or four cases with a total of ten cases. The details of these ten cases and their corresponding switch settings and PE states are shown in Figure 6.2. For each case there may be \( k - 1 \) PEs bypassed between two active PEs. The maximal value of \( k - 1 \) is limited by the delay allowed between two active PEs. The transformation algorithm generates the characteristic matrix of the target array based on the error matrix and the logic matrix. The three output matrices, the plain-switch matrix, the mix-
I. Relation in logic matrix

A. Row mapping group

(1) Case $R_1$ ($c = a$, $d - b = k \geq 1$)

(2) Case $R_2$ ($c - a = l \geq 1$, $d - b = k \geq 1$)

(3) Case $R_3$ ($c - a = l \leq -1$, $d - b = k \geq 1$): similar to Case $R_2$

B. Column mapping group

(4) Case $C_1$ ($c - a = k \geq 1$, $d = b$): similar to Case $R_1$

(5) Case $C_2$ ($c - a = k \geq 1$, $d - b = l \geq 1$): similar to Case $R_2$

(6) Case $C_3$ ($c - a = k \geq 1$, $d - b = l \leq -1$): similar to Case $R_2$

C. Boundary mapping group

(7) Case $B_1$ ($b = k \neq 1$)

(8) Case $B_2$ ($b = k \neq N$): similar to Case $B_1$

(9) Case $B_3$ ($a = k \neq 1$): similar to Case $B_1$

(10) Case $B_4$ ($a = k \neq M$): similar to Case $B_1$

Figure 6.2. Pictorial view of the reconfiguration pattern mapping.
switch matrix, and the PE matrix are derived from the characteristic matrix. These output matrices contain the switch settings for the switches in the vertical link tracks and the switches in the vertical PE tracks, and the PE bypassing information. These matrices need to be converted into binary before they can be used as the input data for VHDL simulation to reconfigure the VHDL-based host array into the target array.

6.2.1. Simulation Set-up

The reconfiguration program and the transformation program are implemented in C on a Sun 3/260 workstation. The conversion of an example host array <6, 6, 1, 1, 2, 0, 0> into a target array <6, 6, 1, 1, 2, 5, 4> is used to illustrate the reconfiguration process. Figure 6.3 shows the symbolic representation of the reconfiguration process for this example. Figure 6.3(a) is the error matrix of the defective host array. The first row in the error matrix represents the size of the host array. The rest of the rows represents the locations of faulty PEs. Figure 6.3(b) is the logic matrix after executing the reconfiguration program. Figure 6.3(c) is the characteristic matrix of the target array after executing the transformation program. The meaning of symbols in the characteristic matrix has been described in Figure 3.2. Figure 6.4 shows the graphical representation of the reconfiguration process for the example array where only active links are retained for clarity. After the first two stages of the reconfiguration process are executed, the Intermetrics Standard VHDL 1076 Support Environment is used to simulate the third stage of the reconfiguration process and matrix multiplications. To demonstrate how to validate the reconfiguration process and the functionality of the target array, an example matrix multiplication shown in Figure 6.5 is performed. The default values of the generic parameters (as shown in Figure 3.5) are used unless they are particularly mentioned. The VHDL simulation process
Figure 6.3. Symbolic representation of the reconfiguration process.
Figure 6.4. Graphical representation of the reconfiguration process.
includes the third stage of the reconfiguration process and a matrix multiplication, and is divided into five parts:

(1) transmitting the plain-switch matrix and the mix-switch matrix from the north side,
and the PE matrix (which initially contains the control data to bypass all the PEs) from the west side of the array to proceed to reconfiguration,

(2) transmitting the multiplicand matrix \([W]\) from the west side of the array via the \(X\) input data paths to the array and storing each element of the matrix in the \(W\) data register of the corresponding PE,

(3) transmitting the PE matrix to reconfigure the host array into the target array,

(4) transmitting the zeros from the west side of the array via the \(X\) input data paths and the multiplier matrix \([Y]\) from the north side of the array via the \(Y\) input data paths to the array, and starting performing multiplication, and

(5) collecting the multiplication product \([X]\) from the east side of the array via the \(X\) output data paths.

Figure 6.5(a) shows how a multiplicand matrix and a multiplier matrix are transmitted to the array and how the product matrix is obtained. Figure 6.5(b) is an example matrix multiplication used in the VHDL simulation. An input format conversion program is necessary to convert the multiplicand and multiplier matrices into the appropriate formats and an output format conversion program is also needed to convert the binary output data into integers and in the form of a matrix.

### 6.2.2. Experimental Results

Both the \(PE_a\)-based array and the \(PE_b\)-based array described in Chapter 3 are simulated. It is found that the \(PE_a\)-based target array needs to operate at \(PE\_cycle\_time\) of 400 ns, while the \(PE_b\)-based target array can operate at \(PE\_cycle\_time\) of 200 ns. This is because the delay due to the bypass register links in the \(PE_a\)-based target array is larger than that of the bypass
links in the PE\textsubscript{b}-based target array. One way to improve \textit{PE\_cycle\_time} of \textit{PEa} is by adopting only one PE clock and adding some delay registers on certain data paths as suggested in [75].

The simulation results compiled from the output report of the VHDL Report Generator for the \textit{PE\textsubscript{b}}-based array is shown in Table 6.1. Note that at 1262 ns, the values of \textit{X\textsubscript{out}} indicate that the \([W]\) matrix is correctly stored in the \(W\) registers of the array. The output \textit{X\textsubscript{out}} is available at an interval of 200 ns. The simulation results verify the correctness of the reconfiguration process and the expected functionality of the target array. It is observed that it takes 13 switch clock cycles to reconfigure the array, 4 PE clock cycles to transmit \([W]\) values, 6 control clock cycles to transmit the PE matrix, and 13 PE clock cycles to perform the matrix multiplication. These results match the theoretical analysis of \(\text{max}(2M+1)SW\_cycle\_time, N\cdot\text{Ctr\_cycle\_time}\) to reconfigure an array, \(V\) PE clock cycles to transmit \([W]\) values, \(N\) control clock cycles to transmit a PE matrix, and \((2U+V-1)\) PE clock cycles to perform a matrix multiplication.

Table 6.1. Simulation results of the \textit{PE\textsubscript{b}}-based array.

<table>
<thead>
<tr>
<th>Time (ns)</th>
<th>\textit{X\textsubscript{out}}(1)</th>
<th>\textit{X\textsubscript{out}}(2)</th>
<th>\textit{X\textsubscript{out}}(3)</th>
<th>\textit{X\textsubscript{out}}(4)</th>
<th>\textit{X\textsubscript{out}}(5)</th>
<th>Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>\text{Start reconfiguration}</td>
</tr>
<tr>
<td>402</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>\text{Start transmitting ([W])}</td>
</tr>
<tr>
<td>1262</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>\text{Start transmitting \textit{PE} matrix}</td>
</tr>
<tr>
<td>1402</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>\text{Start multiplication}</td>
</tr>
<tr>
<td>2350</td>
<td>144</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>\text{\textit{X\textsubscript{11}}}</td>
</tr>
<tr>
<td>2550</td>
<td>158</td>
<td>178</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>\text{\textit{X\textsubscript{12}, X\textsubscript{21}}}</td>
</tr>
<tr>
<td>2750</td>
<td>172</td>
<td>196</td>
<td>212</td>
<td>0</td>
<td>0</td>
<td>\text{\textit{X\textsubscript{13}, X\textsubscript{22}, X\textsubscript{31}}}</td>
</tr>
<tr>
<td>2950</td>
<td>186</td>
<td>214</td>
<td>234</td>
<td>246</td>
<td>0</td>
<td>\text{\textit{X\textsubscript{14}, X\textsubscript{23}, X\textsubscript{32}, X\textsubscript{41}}}</td>
</tr>
<tr>
<td>3150</td>
<td>200</td>
<td>232</td>
<td>256</td>
<td>272</td>
<td>280</td>
<td>\text{\textit{X\textsubscript{15}, X\textsubscript{24}, X\textsubscript{33}, X\textsubscript{42}, X\textsubscript{51}}}</td>
</tr>
<tr>
<td>3350</td>
<td>0</td>
<td>250</td>
<td>278</td>
<td>298</td>
<td>310</td>
<td>\text{\textit{X\textsubscript{25}, X\textsubscript{34}, X\textsubscript{43}, X\textsubscript{52}}}</td>
</tr>
<tr>
<td>3550</td>
<td>0</td>
<td>0</td>
<td>300</td>
<td>324</td>
<td>340</td>
<td>\text{\textit{X\textsubscript{35}, X\textsubscript{44}, X\textsubscript{53}}}</td>
</tr>
<tr>
<td>3750</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>350</td>
<td>370</td>
<td>\text{\textit{X\textsubscript{45}, X\textsubscript{54}}}</td>
</tr>
<tr>
<td>3950</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>400</td>
<td>\text{\textit{X\textsubscript{55}}}</td>
</tr>
</tbody>
</table>
The $PE_b$-based array has the same cycle time of 200 $ns$ (or 5 $MHz$ clock) as the Warp processor array which is a programmable linear systolic array and the Hadamard transform chip which contains eight PEs [17]. Notice that the performance measure for the $PE_b$-based array is done at the processor level, while for the Warp processor array and the Hadamard transform chip it is done at the realization level. To measure the efficiency of the VAR environment, the run time of the first two stages of the reconfiguration process and the VHDL simulation process for the $PE_a$-based and $PE_b$-based arrays is shown in Table 6.2. Note that the third stage of the reconfiguration process is part of the VHDL simulation process. It is observed that the total run time is dominated by the reconfiguration time and the transformation time. The reconfiguration time and the transformation time for larger arrays will be further discussed in the next section. The reason we have a fast VHDL simulation process is due to the simple designs of PE and switch control lines which reduce the actual reconfiguration time of PEs and switches as well as fast matrix multiplications for small array sizes.

6.3. Evaluation and Discussion

The work in [76] emphasized the comparison of algorithm-based fault-tolerant WSI architectures. Other evaluation approaches using different evaluation criteria can be found in

<table>
<thead>
<tr>
<th>Array</th>
<th>Reconfiguration</th>
<th>Transformation</th>
<th>VHDL simulation process</th>
</tr>
</thead>
<tbody>
<tr>
<td>$PE_a$-based</td>
<td>0.014 $s$</td>
<td>0.012 $s$</td>
<td>343.6 $\mu s$</td>
</tr>
<tr>
<td>$PE_b$-based</td>
<td>0.014 $s$</td>
<td>0.012 $s$</td>
<td>171.7 $\mu s$</td>
</tr>
</tbody>
</table>
The figures of merit are described first. To evaluate the efficiency of redundancy and reconfiguration algorithms as well as the quality of target arrays, host arrays with various sizes are simulated and evaluated by using these figures of merit. Simulation results for both random and clustered faults are discussed in Sections 6.3.2 and 6.3.3 respectively.

6.3.1. Figures of Merit

The evaluation criteria for a reconfigurable architecture are based on the following figures of merit: survival probability ($S_p$), locality ($L_A$), maximal interconnection length ($MIL$), hardware overhead ($O_h$), area overhead ($O_a$), yield ($Y_m$), reliability ($R_A$), utilization ($U_r$), and run time ($T_r$, $T_i$) of reconfiguration and transformation algorithms respectively. $S_p$ indicates the reconfiguration capability of a reconfiguration algorithm under different fault distribution patterns. $L_A$, $MIL$, $O_h$, and $O_a$ can be indices of the quality of a target array. $L_A$ ($\geq 1$) indicates the average wire length of an interconnection link. Although array throughput is determined by $MIL$, $L_A$ can tell us the quality of a target array. The larger $L_A$ is, the longer the average wire length of an interconnection link will be. It also means more switches along an interconnection link. As a result, larger $L_A$ implies a higher failure rate of an interconnection link. $MIL$ ($\geq 1$) implies the critical interconnection delay in a target array. A larger $MIL$ implies longer clock cycle time and lower array throughput. $O_h$ can be an index of hardware complexity. Two architectures may have the same area overhead, but one is more complex than the other in terms of the amount of active circuitry. The value of $O_a$ affects the manufacturing yield $Y_m$ significantly. $R_A$ shows the effect of redundancy on array reliability. The mathematical expressions of the figures of merit are derived as follows.
The interconnection length ($IL_{i,j,k,l}$) between PE($i, j$) and PE($k, l$) (or I/O port($k, l$)) in a target array $<M, N, T_h, T_v, S, U, V>$ is defined, in terms of the Manhattan distance with the consideration of the numbers of horizontal and vertical link tracks, as follows:

$$IL_{i,j,k,l} = (T_h+1) |r_i - r_k| + (T_v+1) |c_j - c_l|$$  \hspace{1cm} (1)

where ($r_i, c_j$) is the index in the logic matrix for PE($i, j$) in the target array and ($r_k, c_l$) is the index in the augmented logic matrix for PE($k, l$) or I/O port($k, l$) that is adjacent to PE($i, j$) in the target array. Note that in most VLSI layouts, all geometries must be Manhattan geometries, which means that their edges are parallel to the x and y axes \[79\]. The mathematical definitions of the rest of the figures of merit are either self-explanatory or derived from $IL$ and previous literatures to fit into the PE-switch lattice model. The locality ($L_A$) of a target array $<M, N, T_h, T_v, S, U, V>$ is defined as the average interconnection length of the target array and is derived as follows:

$$L_A = \frac{\sum_{i,j,k,l} IL_{i,j,k,l}}{I_n}$$  \hspace{1cm} (2)

where $I_n$ is the number of interconnections in a non-defect-tolerant array $<U, V, 0, 0, S, U, V>$, i.e.,

$$I_n = U(V+1)+V(U+1).$$  \hspace{1cm} (3)

The maximal interconnection length (MIL) of a target array $<M, N, T_h, T_v, S, U, V>$ is defined as follows:

$$MIL = \max_{i,j,k,l} (IL_{i,j,k,l})$$  \hspace{1cm} (4)

where ($r_i, c_j$) and ($r_k, c_l$) are the indexes of the augmented logic matrix for any two adjacent PE($i, j$) and PE($k, l$), or a PE($i, j$) and its adjacent I/O port($k, l$) in the target array. Note that an interconnection which is an MIL in an array is the critical path of the array. The utilization ($U_r$) of a reconfiguration algorithm is defined as the fraction of good PEs in a host array $<M, N, T_h, T_v, S, U, V>$. 

used in a target array \(<M, N, T_h, T_v, S, U, V>\), i.e.,

\[ U_r = \frac{UV}{MN - F} \times 100\% \]  \(5\)

where \(F\) is the number of faulty PEs in the host array. The hardware overhead \((O_h)\) of a PE-switch array \(<M, N, T_h, T_v, S, U, V>\) is defined as follows:

\[ O_h = \frac{S_n + \beta (MN - UV)}{\beta UV} \times 100\% \]  \(based on the degradation approach) \(6\)

or

\[ O_h = \frac{S_n + \beta P}{\beta (MN - P)} \times 100\% \]  \(based on the redundancy approach) \(7\)

where \(\beta\) is the manufacturing cost ratio of a PE to a switch; \(S_n\) is the number of switches; and \(P = RN + CM - RC\) is the number of extra PEs with \(R\) \((C)\) being the number of extra PE rows \((columns)\). Equations \((6)\) and \((7)\) are equivalent if \(UV = MN - P\). To attain the value of \(O_h\), the number of switches used in the array should be determined first. The number of switches in a type-I reconfigurable array architecture is

\[ S_n = T_h (M - 1)N + T_v (N - 1)M \]  \(8\)

and that in a type-2 reconfigurable array architecture is

\[ S_n = T_h (M + 1)N + T_v (N + 1)M + T_h T_v (M + 1)(N + 1) \]  \(9\)

The area overhead \((O_a)\) of a PE-switch array is derived as follows. Assuming the width of a switch is \(\omega \lambda\), where \(\lambda\) is the length unit. The area \((A_o)\) of a non-defect-tolerant array \(<U, V, 0, 0, S, U, V>\) is equal to

\[ A_o = UV \delta^2 \omega^2 \lambda^2 \]  \(10\)

where \(\delta\) is the width ratio of a PE to a switch. The area of a PE-switch array \(<M, N, T_h, T_v, S, U, V>\) is

\[ A_m = (T_h (M - 1) + \delta M)(T_v (N - 1) + \delta N) \omega^2 \lambda^2 \]  \(based on the type-1 array) \(11\)

\[ A_m = (T_h (M + 1) + \delta M)(T_v (N + 1) + \delta N) \omega^2 \lambda^2 \]  \(based on the type-2 array) \(12\)

Therefore, the area overhead is equal to
The manufacturing yield \( Y_m \) of a PE-switch array \(<M, N, T_h, T_v, S, U, V>\) based on the negative binomial distribution [20] is

\[
Y_m = P(X \leq S_{PE}) = \sum_{k=0}^{S_{PE}} \frac{\Gamma(\alpha+k)(A_mD/\alpha)^k}{k!\Gamma(\alpha)(1+A_mD/\alpha)^{\alpha+k}} S_p(k)
\]

where \( S_{PE} = MN - UV \) is the number of extra PEs; \( \alpha \) is the clustering parameter; \( D \) is the defect density; and \( S_p(k) \) is the probability that the reconfiguration algorithm can reconfigure a faulty host array into the desired target array given \( k \) faulty PEs. The value of \( S_p(k) = \frac{S_k}{T} \) can be attained by the Monte Carlo simulation, where \( S_k \) is the number of fault patterns with \( k \) faulty PEs that can be reconfigured successfully and \( T \) is the total number of fault patterns generated. The manufacturing yield \( Y_o \) of a non-defect-tolerant array \(<U, V, 0, 0, S, U, V>\) obtained by setting \( S_{PE} = 0 \) in Equation (14) is

\[
Y_o = (1+A_mD/\alpha)^{-\alpha}.
\]

Assume that the reliability \( R_{PE}(t) \) of a PE is exponentially distributed with a failure rate \( \lambda_{PE} \), i.e., \( R_{PE}(t) = e^{-\lambda_{PE} t} \). The reliability \( R_A(t) \) of a PE-switch array \(<M, N, T_h, T_v, S, U, V>\) is

\[
R_A(t) = \sum_{k=0}^{S_{PE}} \left( \begin{array}{c} MN \bigg) \end{array} \right)^k S_p(k)
\]

and the reliability \( R_o(t) \) of a non-defect-tolerant array \(<U, V, 0, 0, S, U, V>\) is

\[
R_o(t) = (R_{PE}(t))^{UV}.
\]

6.3.2. Simulation with Random Fault Distribution

Three host arrays, \(<27, 27, 1, 1, 1, 25, 25>, <22, 22, 1, 1, 1, 20, 20>, \) and \(<17, 17, 1, 1, 1, 15, 15>, \) are simulated by using the reconfiguration algorithm \( RR \). Figure 6.6(a) and (b) show
the relationship of reconfiguration time and transformation time with respect to fault sizes. The reconfiguration time is less than 0.2 seconds in most cases. The time needed by transformation is also small (≤ 0.2 seconds) and independent of fault sizes. The utilization is an important index for a reconfiguration algorithm based on the degradation approach. Figure 6.6(c) shows the relationship between utilization and fault sizes. The utilization for these arrays is at least 80% with less than 30 faults. The utilization decreases if the fault size increases. Higher utilization usually results in a higher survival probability. Figure 6.6(d) shows the survival probability of these three arrays. It indicates that the larger the array, the higher the survival probability for a fault size. To study the effect of redundancy on the survival probability, the following host arrays are simulated by using algorithm RR: <21, 20, 1, 1, 1, 20, 20>, <21, 21, 1, 1, 1, 20, 20>, <22, 21, 1, 1, 1, 20, 20>, and <22, 22, 1, 1, 1, 20, 20>. That is, the number of extra rows/columns is in the range from 1 to 4. As expected, Figure 6.6(e) shows that the larger the redundancy, the higher the survival probability. This figure can help the designers determine the appropriate redundancy based on the requirement of $S_p$ and possible fault sizes based on manufacturing yield data. To determine if the amount of redundancy is appropriate, several parameters need to be evaluated. Table 6.3 shows the relationships among $S_p$, $L_A$, $MIL$, $O_h$, $O_a$, and $Y_m$ with respect to different amounts of redundancy by using algorithm RR. The following values are assumed in calculating overhead and yield: $\beta = 8^2$, $\omega = 48$, $\lambda = 1.5 \mu m$ for a 3 $\mu m$ technology, $\alpha = 2$, and $D = 2.5/cm^2$. Note that $S_p$ is the average of the survival probabilities with the fault sizes up to 35. The yield is increasingly enhanced by the extra rows/columns with the penalty of larger $L_A$, $MIL$, $O_h$, and $O_a$. The redundancy should be evenly distributed on rows and columns if the target array is a square array. Table 6.3 also also shows the effect of $\delta$ on yield. The yield decreases if $\delta$ increases. That is, arrays with larger PEs result in a larger wafer area, and thus tend to have lower yield.
Figure 6.6. Array evaluation.
6.3.3. Simulation with Clustered Fault Distribution

Manufacturing defect clustering occurs on a wafer [20, 80]. Therefore, in addition to evaluating the effect of random faults, we also study the effect of clustered faults on reconfigurable arrays. We apply the method in [8] which is based on [81] to generate clustered faults. The generation of clustered faults is controlled by two parameters, $\alpha_1$ and $\alpha_2$, where $\alpha_1$ is the probability of a PE to be faulty at the initial fault generation stage and $\alpha_2$ is the clustering parameter. The clustered fault generator first generates faults with the probability $\alpha_1$ for each PE. Based on this fault pattern, the generator converts a non-faulty PE$(i, j)$ to a faulty PE
Table 6.3. Redundancy evaluation under the random fault distribution.

<table>
<thead>
<tr>
<th>Host array</th>
<th>$S_p$</th>
<th>$L_A$</th>
<th>MIL</th>
<th>$\delta$</th>
<th>$O_h(%)$</th>
<th>$O_a(%)$</th>
<th>$Y_m(%)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;20, 20, 0, 0, 1, 20, 20&gt;</td>
<td>0.00</td>
<td>1.00</td>
<td>1.00</td>
<td>10</td>
<td>0.00</td>
<td>0.00</td>
<td>7.75</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>0.00</td>
<td>0.00</td>
<td>0.77</td>
<td>0.77</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;21, 20, 1, 1, 1, 20, 20&gt;</td>
<td>0.18</td>
<td>2.18</td>
<td>3.85</td>
<td>10</td>
<td>7.00</td>
<td>25.93</td>
<td>51.66</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>5.50</td>
<td>15.23</td>
<td>11.19</td>
<td>11.19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;21, 21, 1, 1, 1, 20, 20&gt;</td>
<td>0.34</td>
<td>2.21</td>
<td>4.39</td>
<td>10</td>
<td>12.35</td>
<td>32.25</td>
<td>81.50</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>10.78</td>
<td>21.00</td>
<td>25.13</td>
<td>25.13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;22, 21, 1, 1, 1, 20, 20&gt;</td>
<td>0.48</td>
<td>2.28</td>
<td>5.02</td>
<td>10</td>
<td>17.70</td>
<td>38.58</td>
<td>90.82</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>16.05</td>
<td>26.78</td>
<td>37.05</td>
<td>37.05</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;22, 22, 1, 1, 1, 20, 20&gt;</td>
<td>0.62</td>
<td>2.33</td>
<td>5.28</td>
<td>10</td>
<td>23.31</td>
<td>45.20</td>
<td>96.41</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>21.58</td>
<td>32.83</td>
<td>49.82</td>
<td>49.82</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

According to the probability, $\alpha_1 + \alpha_2 \cdot \text{adjacent}(i, j)$, where $\text{adjacent}(i, j)$ is the number of faulty PEs that are adjacent to $\text{PE}(i, j)$. Wrap around is assumed at the array boundaries to determine the value of $\text{adjacent}(i, j)$. Therefore, a boundary PE at one side of the array is assumed to be adjacent to the boundary PE at the other side [55]. Compared with Table 6.3, Table 6.4 shows the figures of merit for redundancy evaluation under the clustered fault distribution ($\alpha_1 = 0.001$) using algorithm $RR$. Table 6.3 and Table 6.4 show that the higher the redundancy, the larger the $S_p$, $L_A$, and $MIL$. We can see that the arrays under the clustered fault distribution with $\alpha_2 = 0.01$ have lower yield than the arrays under the random fault distribution. However, increasing the degree of clustering with $\alpha_2 = 0.001$ (i.e. $\alpha_2$ is decreasing) tends to increase $S_p$ and thus enhance array yield. Table 6.4 also shows that $S_p$ increases if $\alpha_2$ decreases. That is, after increasing the degree of clustering to a point, arrays with the clustered fault distribution will have higher yield than arrays with the random fault distribution. This is why the clustered faults have to be considered. Therefore, inclusion of clustering in redundancy yield calculation is of considerable importance [80].
Table 6.4. Redundancy evaluation under the clustered fault distribution.

<table>
<thead>
<tr>
<th>Host array</th>
<th>$S_p$</th>
<th>$L_A$</th>
<th>MIL</th>
<th>$\delta$</th>
<th>$O_A(%)$</th>
<th>$O_{A}(%)$</th>
<th>$Y_m(%)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;21, 20, 1, 1, 1, 20, 20&gt;</td>
<td>0.13</td>
<td>2.12</td>
<td>3.75</td>
<td>10</td>
<td>7.00</td>
<td>25.93</td>
<td>39.52</td>
</tr>
<tr>
<td>($\alpha_2=0.01$)</td>
<td>20</td>
<td>5.50</td>
<td>15.23</td>
<td>7.28</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;21, 20, 1, 1, 1, 20, 20&gt;</td>
<td>0.17</td>
<td>2.19</td>
<td>3.83</td>
<td>10</td>
<td>7.00</td>
<td>25.93</td>
<td>49.83</td>
</tr>
<tr>
<td>($\alpha_2=0.001$)</td>
<td>20</td>
<td>5.50</td>
<td>15.23</td>
<td>10.51</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;21, 21, 1, 1, 20, 20&gt;</td>
<td>0.32</td>
<td>2.15</td>
<td>4.48</td>
<td>10</td>
<td>12.35</td>
<td>32.25</td>
<td>79.88</td>
</tr>
<tr>
<td>($\alpha_2=0.01$)</td>
<td>20</td>
<td>10.78</td>
<td>21.00</td>
<td>24.01</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;21, 1, 1, 1, 20, 20&gt;</td>
<td>0.34</td>
<td>2.20</td>
<td>4.54</td>
<td>10</td>
<td>12.35</td>
<td>32.25</td>
<td>81.82</td>
</tr>
<tr>
<td>($\alpha_2=0.001$)</td>
<td>20</td>
<td>10.78</td>
<td>21.00</td>
<td>25.75</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;22, 21, 1, 1, 20, 20&gt;</td>
<td>0.43</td>
<td>2.24</td>
<td>4.91</td>
<td>10</td>
<td>17.70</td>
<td>38.58</td>
<td>86.50</td>
</tr>
<tr>
<td>($\alpha_2=0.01$)</td>
<td>20</td>
<td>16.05</td>
<td>26.78</td>
<td>32.84</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;22, 1, 1, 1, 20, 20&gt;</td>
<td>0.44</td>
<td>2.25</td>
<td>4.68</td>
<td>10</td>
<td>17.70</td>
<td>38.58</td>
<td>88.39</td>
</tr>
<tr>
<td>($\alpha_2=0.001$)</td>
<td>20</td>
<td>16.05</td>
<td>26.78</td>
<td>33.81</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;22, 22, 1, 1, 20, 20&gt;</td>
<td>0.60</td>
<td>2.27</td>
<td>5.46</td>
<td>10</td>
<td>23.31</td>
<td>45.20</td>
<td>95.57</td>
</tr>
<tr>
<td>($\alpha_2=0.01$)</td>
<td>20</td>
<td>21.58</td>
<td>32.83</td>
<td>45.57</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;22, 22, 1, 1, 20, 20&gt;</td>
<td>0.67</td>
<td>2.31</td>
<td>5.18</td>
<td>10</td>
<td>23.31</td>
<td>45.20</td>
<td>97.00</td>
</tr>
<tr>
<td>($\alpha_2=0.001$)</td>
<td>20</td>
<td>21.58</td>
<td>32.83</td>
<td>50.88</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

To illustrate the effect of the numbers of link tracks and clustered faults as well as random faults on array yield by using the reconfiguration algorithms RR, BR, and BB respectively, the host arrays, <22, 22, 1, 1, 1, 20, 20>, <22, 22, 1, 0, 1, 20, 20>, and <22, 22, 0, 0, 1, 20, 20>, are simulated. Table 6.5 shows the simulation results, where $\alpha_1$ is set to 0.001. A more complex switching mechanism results in larger $S_p$, $L_A$, and MIL as demonstrated in Table 6.5. Under the clustered fault distribution, if the degree of clustering is increased, the arrays for algorithms RR and BR have higher yield, but the arrays for algorithm BB have lower yield. Note that for the arrays using algorithm BB, their $S_p$ and $L_A$ stay the same even when $\alpha_2$ decreases. So algorithms RR and BR perform better for arrays with clustered faults. Algorithm BB has smaller overall yield because it does not have link tracks to provide row or column rerouting capability. One surprising result is that the arrays for algorithm BR have comparable
Table 6.5. Reconfiguration algorithm evaluation.

<table>
<thead>
<tr>
<th>Alg.</th>
<th>Host array</th>
<th>$S_p$</th>
<th>$L_A$</th>
<th>MIL</th>
<th>$\delta$</th>
<th>$O_h(%)$</th>
<th>$O_o(%)$</th>
<th>$Y_m(%)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>RR</td>
<td>&lt;22, 22, 1, 1, 1, 20, 20&gt; (random)</td>
<td>0.62</td>
<td>2.33</td>
<td>5.28</td>
<td>10</td>
<td>23.31</td>
<td>45.20</td>
<td>96.41</td>
</tr>
<tr>
<td></td>
<td>&lt;22, 22, 1, 1, 1, 20, 20&gt; ($a_2 = 0.01$)</td>
<td>0.60</td>
<td>2.27</td>
<td>5.46</td>
<td>10</td>
<td>23.31</td>
<td>45.20</td>
<td>95.57</td>
</tr>
<tr>
<td></td>
<td>&lt;22, 22, 1, 1, 1, 20, 20&gt; ($a_2 = 0.001$)</td>
<td>0.67</td>
<td>2.31</td>
<td>5.18</td>
<td>10</td>
<td>23.31</td>
<td>45.20</td>
<td>97.00</td>
</tr>
<tr>
<td>BR</td>
<td>&lt;22, 22, 1, 0, 1, 20, 20&gt; (random)</td>
<td>0.61</td>
<td>1.72</td>
<td>4.28</td>
<td>10</td>
<td>22.16</td>
<td>32.55</td>
<td>96.71</td>
</tr>
<tr>
<td></td>
<td>&lt;22, 22, 1, 0, 1, 20, 20&gt; ($a_2 = 0.01$)</td>
<td>0.54</td>
<td>1.70</td>
<td>4.47</td>
<td>10</td>
<td>22.16</td>
<td>32.55</td>
<td>94.71</td>
</tr>
<tr>
<td></td>
<td>&lt;22, 22, 1, 0, 1, 20, 20&gt; ($a_2 = 0.001$)</td>
<td>0.60</td>
<td>1.69</td>
<td>4.15</td>
<td>10</td>
<td>22.16</td>
<td>32.55</td>
<td>96.63</td>
</tr>
<tr>
<td>BB</td>
<td>&lt;22, 22, 0, 0, 1, 20, 20&gt; (random)</td>
<td>0.19</td>
<td>1.05</td>
<td>1.91</td>
<td>10</td>
<td>21.00</td>
<td>21.00</td>
<td>58.70</td>
</tr>
<tr>
<td></td>
<td>&lt;22, 22, 0, 0, 1, 20, 20&gt; ($a_2 = 0.01$)</td>
<td>0.19</td>
<td>1.05</td>
<td>2.06</td>
<td>10</td>
<td>21.00</td>
<td>21.00</td>
<td>60.49</td>
</tr>
<tr>
<td></td>
<td>&lt;22, 22, 0, 0, 1, 20, 20&gt; ($a_2 = 0.001$)</td>
<td>0.19</td>
<td>1.05</td>
<td>1.83</td>
<td>10</td>
<td>21.00</td>
<td>21.00</td>
<td>59.02</td>
</tr>
</tbody>
</table>

yield with the arrays for algorithm RR, although algorithm RR is more flexible. Arrays for algorithm RR have a little higher $S_p$, but this is offset by the larger area due to more switches than arrays for algorithm BR. This shows that it is not necessarily better to have more switches since the extra switches would increase the area without increasing the size of the target array. Therefore, for certain applications, one may want to choose the redundancy strategy in algorithm BR which has smaller $L_A$, MIL, $O_h$, and $O_o$. Figure 6.7 illustrates the relationship of reliability with respect to time for various reconfigurable arrays as well as the non-defect-tolerant array under the clustered fault distribution ($\alpha_1 = \alpha_2 = 0.001$). The PE failure rate is assumed to be $\lambda_{PE} = 0.1$ failures per unit time. Note that the array <22, 21, 1, 1, 1, 20, 20> has only a
slightly higher reliability than the array <21, 21, 1, 1, 1, 20, 20>, although the former has an extra row. However, without redundancy the reliability of the non-defect-tolerant array will fall off rapidly. The above simulation results demonstrate that by evaluating the figures of merit through simulation, we can choose a better combination of the redundancy scheme, switching mechanism, and diagnosis and reconfiguration algorithms to design a reconfigurable array based on actual manufacturing yield data.

6.4. Summary

In this chapter, we have presented the simulation and evaluation processes for defect-tolerant two-dimensional arrays in the VAR system. The simulation of the reconfiguration process in VAR is implemented by interfacing the reconfiguration program, the transformation program, and the VHDL description of the array. Extensive simulation has been performed and experimental results are obtained which indeed demonstrate the effectiveness of our approach. The VAR system will be able to assist the designers to simulate and evaluate different design
strategies for architectural improvements.
CHAPTER 7.

CONCLUSIONS

7.1. Summary

In this dissertation, we have focused on the four issues in the VAR (VHDL-based Array Reconfiguration) system: (1) the development of a CAD framework for reconfigurable architectures, (2) the development of an array model, and its VHDL description and simulation, (3) the development of efficient fault diagnosis techniques, and (4) the development of a systematic method for evaluating architectures and yield. In the following, the research results from these four issues are summarized.

The integrated high-level CAD environment, VAR, for the design, diagnosis, reconfiguration, simulation, and evaluation of defect tolerant VLSI or WSI (wafer scale integration) parallel architectures has been presented. The simulation of the reconfiguration process in VAR is implemented by interfacing the reconfiguration program, the transformation program, and the VHDL description of the array. Extensive simulation has been performed and experimental results are obtained which indeed demonstrate the effectiveness of our approach. The VAR system will greatly help the designers to evaluate different redundancy strategies, various fault diagnosis and reconfiguration techniques, quality of target arrays, yield, and reliability.

We have presented three fault diagnosis techniques. The first fault diagnosis technique is based on the parallel partition approach. The diagnosability and reconfigurability of an architec-
ture are enhanced by utilizing the bypass register links in each processing element (PE). This approach is applicable to a variety of parallel interconnection networks. Multiple PE-faults in these architectures can be detected and located. The minimum speed-up of this approach over the serial testing is $O(|V|^{1/n})$, where $|V|$ is the number of PEs. With the rapid advancement of integrated circuit technologies, the sizes of VLSI/WSI parallel architectures will continue to grow and the advantages of this approach will be more significant. This methodology is very cost-effective because very little extra hardware is added for diagnosis and the extra hardware can also benefit reconfiguration.

The second fault diagnosis technique is based on the self-comparison approach. Each PE is designed to be easily diagnosable by including extra circuits. The approach is applicable to any PE structures in various parallel architectures. The multiple PE-faults in an architecture can be detected and located. The time complexity of the algorithm is $O(N)$, where $N$ is the number of test patterns and is independent of the size of an architecture. The hardware overhead is not significant. The design based on this approach is also suitable for concurrent error detection and location.

The third fault diagnosis technique is also based on the parallel partition approach. It is an optimal group diagnosis procedure for defect tolerant VLSI or WSI parallel architectures. The optimization criterion is the minimization of the accumulated diagnosis cost. A $(t+1)$-ary decision tree is used to model an optimal group diagnosis problem. The complexity of dealing with a $(t+1)$-ary decision tree is then reduced to that of handling a binary decision tree. A block-walking representation is then used to denote a binary decision tree corresponding to the static optimal group diagnosis, and thus reduces the space complexity from exponential complexity to polynomial complexity. Properties related to the group diagnosis procedures and
binary decision trees have been derived. The diagnosis and reconfiguration processes are integrated to take full advantage of the optimal group diagnosis procedure. The taxonomy and complexity of various diagnosis strategies are analyzed and compared. Extensive simulation has been performed and experimental results are evaluated to demonstrate the effectiveness of the optimal group diagnosis procedure.

The problems in the development of large area VLSI and WSI may be solved via defect tolerance techniques. A wafer manufactured with a number of PEs will likely contain a certain number of faulty elements as soon as it comes off the production line. If the design depends upon all the elements being functional to be useful, we might never get a working device. Therefore, the design must be performed such that the faulty elements within the architecture can be switched out, and the fault-free spare elements can be switched in to achieve an operational architecture. But spare allocation and reconfiguration are not possible without the detection and location of faulty elements. Fault diagnosis is a prerequisite to successful reconfiguration and is extremely difficult in VLSI/WSI due to low controllability and observability. This research has proposed three novel fault diagnosis techniques and a CAD framework for defect tolerant VLSI/WSI parallel architectures. Our work will lead to better understanding of the capabilities and limitations of the new yield enhancement and architectural improvements techniques. The results from this research will help to make the application of large area VLSI and WSI technologies possible in the near future.

7.2. Suggested Future Research

In this dissertation work, we have concentrated on the developments of a CAD framework, the VHDL modeling method, fault diagnosis techniques, and the simulation and
evaluation of the reconfiguration process for defect-tolerant VLSI/WSI parallel architectures in the VAR system. Suggested future research issues include: (1) simulation and evaluation of the fault diagnosis process, (2) optimization in terms of architectural designs, fault diagnosis techniques, and reconfiguration techniques, and (3) interface with a VHDL synthesis system for the layout generation.
REFERENCES


139


