WARWICK
THE UNIVERSITY OF WARWICK

A Thesis Submitted for the Degree of PhD at the University of Warwick

Permanent WRAP URL:
http://wrap.warwick.ac.uk/109958

Copyright and reuse:
This thesis is made available online and is protected by original copyright.
Please scroll down to view the document itself.
Please refer to the repository record for this item for information to help you to cite it.
Our policy information is available from the repository home page.

For more information, please contact the WRAP Team at: wrap@warwick.ac.uk

warwick.ac.uk/lib-publications

THE BRITISH LIBRARY DOCUMENT SUPPLY CENTRE

TITLE

Self-organising Techniques
for Tolerating Faults in
2-Dimensional Processor Arrays

AUTHOR

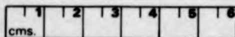
Richard Anthony Evans

INSTITUTION
and DATE

University of Warwick
1988

Attention is drawn to the fact that the copyright of this thesis rests with its author.

This copy of the thesis has been supplied on condition that anyone who consults it is understood to recognise that its copyright rests with its author and that no information derived from it may be published without the author's prior written consent.



THE BRITISH LIBRARY
DOCUMENT SUPPLY CENTRE
Boston Spa, Wetherby
West Yorkshire
United Kingdom

REDUCTION X 20

CAMERA 8

Self-organising Techniques for Tolerating Faults in 2-Dimensional Processor Arrays

by

Richard Anthony Evans
BSc(Eng), ACGI, CEng, MIEE

PhD Thesis

Submitted to:
Department of Computer Science
University of Warwick

From research carried out at:
The Royal Signals and Radar Establishment
Malvern, Worcestershire

October 1988

To my parents

For their love,
support and encouragement
over many years.

Copyright © Controller HMSO London, 1988

Contents

List of Figures vii

List of Tables x

Summary xi

Acknowledgements xii

1 Introduction, Objectives and Overview 1
- 1.1 Introduction 1
- 1.2 Thesis Objectives 2
- 1.3 Overview of the Thesis 3

2 The Evolution of Parallel Processing 5
- 2.1 The von Neumann Model of Computation 5
  - 2.1.1 Technological Advances 6
  - 2.1.2 The Need for High Performance Computers 7
  - 2.1.3 Changing Attitudes to Computer Design 7
- 2.2 The Introduction of Parallelism 8
  - 2.2.1 Classification of Computer Architectures 9
  - 2.2.2 Pipelined Parallel Architectures 11
- 2.3 The Need for Fault Tolerance 14
  - 2.3.1 Reliability Improvement 14
- 2.4 Wafer Scale Integration 15
- 2.5 Types of Fault Tolerance 16
  - 2.5.1 Time Redundancy 17
  - 2.5.2 Hardware Redundancy 17
  - 2.5.3 Algorithmic Fault Tolerance 18
- 2.6 Scope of the Project 19

3 Fault Distributions in Integrated Circuits 20
- 3.1 Introduction 20
- 3.2 Types of Defect in Integrated Circuits 21

3.3 Random Point Defects 23
3.3.1 Causes of Point Defects 23
3.3.2 Clustering of Point Defects 26
3.3.3 Radial Variations in Point Defect Distributions 28
3.4 Modelling Integrated Circuit Yield 29
3.4.1 Poisson Distribution 31
3.4.2 Compound Poisson Statistics 33
3.4.3 Generalised Negative Binomial Statistics 34
3.5 Implications for Fault Tolerant Techniques 35

4 A Review of Hardware Fault-Tolerance for Processor Arrays 37
4.1 Introduction 37
4.2 Classification of hardware fault tolerance schemes 38
4.3 Switch Organisation and Configuration Schemes 39
4.3.1 Nodal Fault Tolerance 39
4.3.2 Row or Column Replacement Schemes 44
4.3.3 Hierarchical Fault Tolerance 48
4.3.4 Row Generation Schemes 49
4.3.5 Global Organisation 51
4.4 Switch Implementations 52
4.4.1 Hard Configurable Schemes 53
4.4.2 Firm Configurable Switching Schemes 56
4.4.3 Soft-Configurable Switching Schemes 58
4.4.4 Vote-Configurable Switching Schemes 59
4.4.5 Self-Organising Switching Schemes 59
4.5 WSI Demonstrators 59
4.5.1 Trilogy 60
4.5.2 Anamartic and the Solid State Disk Memory 60
4.5.3 MIT and Lincoln Laboratory 63
4.5.4 GTE Laboratory 63
4.6 Conclusions 63

5 Self-Organising Algorithms for Two-Dimensional Processor Arrays 65
5.1 Introduction 65
5.2 Definitions 66
5.3 Algorithm 1: *WINNER* in One Dimension 68
5.3.1 Self Organisation 71
5.3.2 Control Circuitry 72
5.3.3 Array Boundary Conditions 75
5.3.4 Interaction of REQuest and AVAILability Signals . . . 76
5.3.5 Serial Description of *WINNER* Operation 77
5.3.6 5-Neighbour *WINNER* algorithm 79

5.4 Algorithm 2: *WINNER* In Two Dimensions . . . 80
5.4.1 Double Site Condition . . . 82
5.4.2 Crossovers . . . 85
5.5 Advantages of the *WINNER* Approach . . . 87
5.6 Concluding Remarks . . . 87

6 Performance of the WINNER Algorithm 89
6.1 Introduction . . . 89
6.2 Simulation . . . 89
6.3 Choice of Language . . . 90
6.4 Simulation Requirements . . . 91
6.4.1 Program Parameters . . . 93
6.4.2 Program Flow Chart . . . 94
6.4.3 Square Arrays . . . 95
6.5 *WINNER* Simulation Results . . . 97
6.5.1 Graphical Presentation of *WINNER* Results . . . 98
6.6 Improving *WINNER* Performance . . . 102
6.6.1 5-Neighbour *WINNER* Algorithm . . . 103
6.6.2 Array Partitioning . . . 104
6.7 Comparison with other Algorithms . . . 109
6.7.1 Bounds on Performance . . . 109
6.7.2 Algorithm Performance Comparisons . . . 111
6.8 Conclusions . . . 113

7 Hardware Implementation of the WINNER algorithm 115
7.1 Introduction . . . 115
7.2 Hardware Requirements of *WINNER* Control Circuitry . . . 116
7.2.1 Gate-Level *WINNER* Control Circuitry . . . 116
7.2.2 Transistor Complexity of Control Circuitry . . . 118
7.3 Input/Output Interface Circuitry . . . 120
7.3.1 Selecting Functional Rows . . . 120
7.3.2 Data Input Circuitry . . . 121
7.3.3 Data Output Circuitry . . . 124
7.4 Column Interconnnection Circuitry in Partitioned Arrays . . . 124
7.5 Simulation of the *WINNER* Hardware . . . 127
7.5.1 The ELLA Hardware Description Language . . . 127
7.5.2 Simulations using ELLA . . . 128
7.6 Hardware for other Configuring Techniques . . . 128
7.6.1 Moore and Mahat's Scheme . . . 128
7.6.2 Sami and Stefanelli's Scheme . . . 129
7.6.3 Comparison of Hardware Requirements . . . 129

8 Self-Testing of Self-Organising Arrays 131
8.1 Introduction 131
8.2 Design for Testability 131
8.2.1 Scan path techniques 132
8.2.2 Level Sensitive Scan Design - LSSD 133
8.3 Self-test techniques 134
8.3.1 Linear Feedback Shift Registers 135
8.3.2 Compression of test results 137
8.3.3 Compression by counting 137
8.3.4 Compression by Recursive Compaction 137
8.3.5 Compaction of Multiple Input Streams 138
8.4 Self testing requirements in *WINNER* 139
8.4.1 Signature Comparison 139

9 Reducing Control Circuit Vulnerability 141
9.1 Introduction 141
9.2 The Ideal Self-Organising Array 142
9.3 Inherent Fault-Tolerance of the Control Circuit 143
9.4 Dual-Rail Implementation of Control Circuitry 143
9.4.1 Fault Detection by Simple Duplication 144
9.4.2 Fault Detection using True and Complement Circuits 146
9.5 Application of Duplicated Circuits to *WINNER* 147
9.5.1 Performance of Duplicated Control Circuitry 149
9.6 Hardware Requirements for Two-rail Implementation 151
9.7 External Testing of the Control Circuitry 151
9.7.1 Control Circuit Testing Strategy 151
9.7.2 Scan Path testing procedure 154
9.7.3 Control Circuitry Fault Masking Procedure 155
9.7.4 Modified Scan path Register 155
9.8 Hardware Complexity of the Scan Path Approach 156
9.9 Other Tests 157
9.9.1 Testing of the Signature Comparator 157
9.9.2 Vertical Bypass Circuitry 160
9.9.3 Horizontal Interconnections 162
9.10 Benefits of the Scan path Test procedure 162
9.11 Comments 163

10 WINNER Demonstrator 164
10.1 The Need for a Demonstrator 164
10.2 Type of Demonstrator 164
10.3 Demonstrator Objectives 165
10.4 Demonstrator Specification 166
10.5 Processor Array Function 167

10.6 Implementation Options 169
10.6.1 LSI Components 169
10.6.2 Custom Chip or Gate Array 169
10.6.3 EPROM Implementation 171
10.7 Demonstrator Circuitry Requirements 171
10.7.1 Circuit for the *WINNER* Cell 173
10.7.2 Other Circuitry 176
10.8 External Testing 179
10.8.1 Computer Interface circuitry 180
10.8.2 Testing Software 180
10.9 Demonstrator Results 180
10.10 Concluding Remarks 184

11 Applications of WINNER **185**
11.1 Introduction 185
11.2 Application to Processor Arrays 185
11.2.1 The Transputer 186
11.2.2 The Distributed Array Processor (DAP) 187
11.2.3 Digital Signal Processing Arrays 188
11.3 Application in Array Implementation 188
11.4 High Availability Systems 189
11.4.1 Special Considerations 189
11.4.2 General Comments 190
11.5 Silicon Hybrids 191
11.5.1 Requirement for Fault Tolerance 192
11.6 Wafer Scale Integration 192

12 Conclusions and Further Work **194**
12.1 Conclusions 194
12.2 Suggestions for Further Work 197
12.2.1 Fabrication of a Monolithic circuit 197
12.2.2 Extension to Tree Structures 197
12.2.3 Fault Tolerant Switching Networks 198

Bibliography **200**

APPENDICES

A WINNER Performance Simulation Program **210**
A.1 Program Overview 210
A.2 Suitability for Other Algorithms 211
A.3 Program Listing 212

B ELLA Simulation Program **219**
B.1 Program Listing 219

C WINNER Demonstrator Test Program **227**
C.1 Program Overview 227
C.2 Program Listing 227

D Published Papers **237**
D.1 Papers Included in the Appendix 237
D.2 Other Publications 238

List of Figures

2.1 Systolic Matrix × Matrix Multiplier 12
2.2 Typical arrangement for algorithmic fault-tolerance. 18

3.1 The main Classes of integrated Circuit Defect. 21
3.2 Types of Integrated Circuit Defect 23
3.3 Sizes of Integrated Circuit Defects 25
3.4 Probability of defect causing a fault 26
3.5 Typical Clustering tendency of defects 27
3.6 Radial yield variation 29
3.7 Chip yield predicted by the Poisson model 32
3.8 Number of faults per chip 33
3.9 Yield predicted by Negative Binomial model 35

4.1 Classification of switching schemes 39
4.2 Classification of switch implementations 40
4.3 Technique of Triple Modular Redundancy 42
4.4 Nodal yield improvement achieved with TMR. 43
4.5 Toleration of TMR voting circuit faults 44
4.6 Fault-tolerant memory 45
4.7 Simple Row-bypass Scheme 47
4.8 Hierarchical fault-tolerance scheme 48
4.9 Row-oriented scheme of Moore and Mahat 50
4.10 A global interconnection scheme 52
4.11 Cross-section of laser link 55
4.12 MNOS transistor. 57
4.13 Spiral technique of Aubusson and Catt 62

5.1 Main components of a *WINNER* cell 67
5.2 Equivalent hexagonal and orthogonal arrays 68
5.3 Array configured using *WINNER* 69
5.4 Schematic *WINNER* cell 72
5.5 Step-by-step array configuration 78
5.6 Generation of REQuest signals 79
5.7 *WINNER* applied in 2 dimensions 81

5.8 *WINNER* cell for 2D configuration 83
5.9 Double-site and crossover modes 84

6.1 Harvest of Hedlund's scheme 93
6.2 Schematic of the table of simulation results. 94
6.3 Characteristic form of the table of results. 95
6.4 Flow-chart of the simulation program. 96
6.5 Table of simulation results 98
6.6 Array yield versus processor yield 100
6.7 Overhead versus processor yield 101
6.8 Array yield versus target array size 103
6.9 5-Neighbour *WINNER* performance 104
6.10 Array yield versus array width 106
6.11 Effect of partitioning on array yield 107
6.12 Array yield versus number of partitions 107
6.13 Upper bounds on configuration performance 111
6.14 Comparison of configuration performance 112

7.1 Gate implementation of control circuitry 117
7.2 CMOS implementation of some common gates 119
7.3 Data input circuitry 122
7.4 Data output circuitry 125
7.5 Intercolumn routing circuitry 126
7.6 Comparison of hardware requirements 130

8.1 The scan-path testing technique. 133
8.2 Block diagram of a typical self-testing system. 135
8.3 A linear feedback shift register 136
8.4 Single-input compaction circuit 138
8.5 Multiple-input compaction circuit 139
8.6 Simple signature comparison method. 140

9.1 Fault detection by simple duplication 145
9.2 True and complement two-rail implementation. 146
9.3 Two-rail input buffer circuits 148
9.4 A two-rail signature comparator. 149
9.5 Percentage of active faults 150
9.6 Scan-path scheme applied to *WINNER* 152
9.7 Scan-path arrangement within a *WINNER* cell 153
9.8 Modified scan-path register 156
9.9 A TMR based signature comparator. 159
9.10 A fully testable signature comparator. 160
9.11 A fully verifiable *WINNER* cell. 161

10.1 4-bit by 4-bit array multiplier 168
10.2 Orthogonally interconnected multiplier 170
10.3 Block diagram of the *WINNER* demonstrator. 172
10.4 Circuit of demonstrator cell 174
10.5 Equivalent circuit of the Control PROM. 175
10.6 Equivalent circuit of the Processor PROM. 176
10.7 Equivalent circuit of the Scan PROM. 177
10.8 Equivalent circuit of data entry PROM 178
10.9 Equivalent circuit of data output PROM 179
10.10 Photograph of complete demonstrator 181
10.11 Single cell of the demonstrator array 182
10.12 Photograph of configured demonstrator array 183

11.1 Schematic view of the Transputer. 186
11.2 Schematic view of the ICL DAP. 187

List of Tables

3.1 Types of Integrated Circuit Defect. 24

4.1 Laser link parameters. 55

5.1 Generation of AVAILability output signals. 74
5.2 Generation of REQuest output signals. 76

9.1 Conversion of two-rail input signals 148

Summary

This thesis is concerned with research into techniques for tolerating the defects which inevitably occur in integrated circuits during processing. The research is motivated by the desire to permit the fabrication of very large ($\gg 1cm^2$) integrated circuits having a viable yield, using standard chip processing lines. Attention is focussed on 2-dimensional arrays of identical processing elements with nearest-neighbour, orthogonal interconnections, and techniques for configuring such arrays in the presence of faults are investigated. In particular, novel algorithms based on the concept of *self-organisation* are proposed and studied in detail. The algorithms involve associating a small amount of control logic with each processing element in the array. The extra logic allows the processing elements to communicate with each other and come to a collective decision about how working processors should best be interconnected. The concept has been studied in considerable depth and the implications of the algorithms in a practical system have been thoroughly considered and demonstrated by construction of a small array at printed circuit board level, complete with software controlled testing procedures.

The thesis can be considered in four main parts as follows. The first part (chapters 1 to 4) starts by presenting the objectives of the research and then motivates it by examining the increasing need for processor arrays. The difficulty of implementing such arrays as monolithic circuits due to integrated circuit defects is then considered. This is followed by a review of published work on hardware fault tolerance for regular arrays of processors. The second part (chapters 5 and 6) is devoted to the concept of *self-organisation* in processor arrays and includes a detailed description and evaluation of the algorithms followed by a comparison with other published techniques. Considerations such as hardware requirements and overheads, reducing the vulnerability of critical circuitry, self-testing, and the construction of the demonstrator are covered in the third part (chapters 7 to 10). The fourth part (chapters 11 and 12) considers potential applications for the research in both monolithic and non-monolithic systems. Finally, the conclusions and some suggestions for further work are presented.

Acknowledgements

The author wishes to acknowledge the support, assistance and encouragement of the following people and institutions throughout the course of the research leading to this thesis:

Dr Tim Thorp, for encouraging me to register as a postgraduate student and for being understanding during the writing of the thesis,

Dr John McWhirter, for his inspiration, and day to day encouragement and support,

Dr Adrian Mears, for suggesting that I register as a postgraduate student,

Professor Graham Nudd, for supervising me throughout the thesis and for his suggestions, encouragement and comments,

Dr John McCanny, for his helpful discussions and comments on the thesis,

Alan Brewer and Gary Ellis, for their hard work in building the demonstrator of a self-organising array,

Kevin Palmer, for helpful discussions and assistance during the design and construction of the demonstrator array,

Dr Ian Proudler, for many useful discussions, particularly concerning two-rail circuit implementations,

My wife Josie, for her love and support throughout this thesis but particularly during the difficult times,

The Computer Science Department of the University of Warwick, for accepting me as a part-time postgraduate student,

The Ministry of Defence (Procurement Executive), for providing full funding for University fees, travel and attendance at conferences as well as for permitting me to submit my research as a PhD thesis.

Chapter 1

Introduction, Objectives and Overview

1.1 Introduction

"In 1959 the number of transistors that would fit on a chip was 1; now it has surpassed 1 million. As material limits are reached, the pace is slowing, but by the year 2000 there will be chips containing 1 billion components".

This quotation by James Meindl (Meindl, 1987) is partly factual and partly speculative, but certainly indicates the scale of the challenge facing integrated circuit researchers and designers over the coming decade. Currently the chips containing 1 million transistors are memory devices consisting of vast numbers of identical storage cells, and are fabricated on highly optimised processing lines and sold in huge volumes. General purpose circuits have not yet reached this complexity in part because they cannot be sold in high enough volumes to sufficiently amortise the design and fabrication costs. The fabrication costs of these very large general purpose circuits are high because their yield is low. If techniques could be found to increase the yield, costs would fall and marketability should significantly increase.

One way to increase chip yield is to reduce the number of defects which occur randomly in the processing line. This is already being done wherever possible, but with continual reductions in device geometries, the task is be-

coming ever more difficult and increased standards of cleanliness are often required simply to maintain existing yields. The net result is that chips above about 1 cm^2 in area are rarely produced commercially.

Reductions in device geometry permit increased numbers of transistors to be placed on a chip of any given area. However, although further reductions are possible it is clear that the rate of reduction is slowing down. This is mainly due to the fact that the equipment and process lines needed to handle the small device structures become almost exponentially more expensive, requiring fundamental changes in lithographic techniques and etching techniques for example.

With this background it is hardly surprising that many researchers have been investigating an alternative technique to enable chip complexity to be significantly increased at reasonable cost. This technique is called *Fault Tolerance* or *Defect Tolerance*. The application of fault tolerant techniques to complex electronic systems has received the attention of researchers since electronics began to be used in safety-critical applications such as the control of aircraft or large industrial plant. These systems start life in a fully functional form but in the event of the in-service failure of a component, can either fail in a safe manner, or ideally, continue their function as if no fault were present. The application of fault tolerant techniques to integrated circuits is in general quite different. In integrated circuits the problem is that a small number of components within a large chip are faulty at the start of the chip's life and these cause the entire chip to be discarded as a failure. The primary aim therefore is to develop the ideas of fault tolerance to be able to tolerate *defects* present in the circuits. The ability of the circuit to tolerate in-service failures in addition to fabrication defects would be a bonus.

1.2 Thesis Objectives

In this thesis we address the problem of design techniques to enable integrated circuits to tolerate fabrication defects. In particular we consider the

implementation of large 2-dimensional arrays of identical processing elements with nearest-neighbour, orthogonal interconnections, in which some of the processing elements may be permanently defective. The objectives of the thesis are as follows:

1. To study existing techniques for fault tolerance or defect tolerance applied to integrated circuit fabrication.
2. To develop novel techniques for tolerating defects based on the concept of *Self-organisation*. Self-organisation implies that a circuit is both able to detect and then automatically configure itself around each defect without any external assistance to produce a fully functional array.
3. To evaluate the performance of the self-organising techniques developed in (2), and to compare them with existing approaches to the configuration problem.
4. To demonstrate the self-organising techniques by constructing a processor array which embodies the concepts.

1.3 Overview of the Thesis

In order to assist the reader, there follows a brief overview of the thesis on a chapter by chapter basis.

Chapters 2 and 3 motivate the work in the remainder of the thesis and together attempt to answer the question: *Why do we need fault tolerance?* Chapter 2 reviews the evolution of computing from single von Neumann machines to the large multi-processor architectures available today and considers some of the reasons for the trend towards parallel processing. Chapter 3 then considers in detail the types of defect which occur in integrated circuits and presents some mathematical models which are commonly used to predict the yield of integrated circuits. Characteristic yield variations over the wafer surface are also discussed.

Chapter 4 reviews the state of the art in techniques suitable for tolerating integrated circuit defects. Schemes for configuring circuits around defects as well as techniques for implementing the switching elements are considered.

Chapter 5 describes the novel self-organising algorithms which have been developed within the project. The reader who is familiar with the background and motivation for fault tolerant techniques can skip straight to this chapter if desired. The self-organising algorithms are evaluated in chapter 6 and then compared with the existing published techniques. The hardware requirements of the self-organising algorithms are considered in chapter 7.

The concept of self-organisation requires that each processor in the array can produce a *go/no-go* signal to indicate whether it is functional or not. This signal is then used in the configuration process. Chapter 8 discusses how this could be achieved by *Self-testing* techniques.

One of the problems in a system designed to automatically tolerate defects is that the circuitry used to carry out the configuration could itself be defective. This problem is addressed in chapter 9 in which two techniques for reducing the vulnerability of the most critical circuits are presented.

Chapters 7 and 10 are both concerned with the hardware required in a self-organising array. Chapter 7 considers the hardware overhead which the self-organising approach will incur for the user, while chapter 10 describes an array which has been built to demonstrate the ideas incorporating the techniques described in chapters 5 and 9.

Finally, in chapters 11 and 12 we consider potential applications of the self-organising concepts other than in integrated circuit fabrication, and then conclude by putting forward some suggestions for further work.

Chapter 2

The Evolution of Parallel Processing

2.1 The von Neumann Model of Computation

For the past 30 years or so the von Neumann model of computation has dominated nearly every aspect of computing from the largest mainframe to the smallest home computer. During this period, however, the physical appearance of von Neumann computers has changed dramatically. In the early days of computing a complete room full of vacuum tube circuitry and many kilowatts of power were required to run a machine of modest capability (by todays standards!). Today, it is possible to integrate a computer of many times this processing power on a single chip and sell a complete machine at such a low price that it has become possible to buy one for a child at Christmas! Machines of this type are of course quite basic computers and their speed of operation is not the highest consideration. At the other extreme, large mainframe machines which serve whole departments or sites on a time-sharing basis are still quite bulky pieces of equipment, but are able to handle many tens or sometimes hundreds of users almost simultaneously.

2.1.1 Technological Advances

Although the von-Neumann model of computing is a simple concept and to some who are working at the frontiers of computing research now appears a little 'old fashioned', it still stands up very well to today's requirements for general purpose computing. This has been made possible by essentially four major technological advances over the years as described in the following paragraph.

The first commercially produced computer, the UNIVAC1, appeared in 1951 and used electronic valves with gate delays of about 1μs as its switching elements (Eckert et al, 1951). Machines of this type have been classed as First Generation computers. In 1960, valves were replaced by solid state devices, in the form of germanium transistors, which had gate delays of approximately 0.25μs - these were Second Generation machines. The first silicon integrated circuits were introduced in about 1965 and contained a few gates per chip operating with a 10ns gate delay and by the mid seventies achieving 1ns gate delays. Third generation machines used these components. In the 1970s, the ability to manufacture integrated circuits became widespread, and their complexity rapidly increased so that by 1980 it became possible to integrate a complete microcomputer onto a single chip - the era of fourth generation computing had begun.

The microprocessor was largely reponsible for two major trends in computing in the 1980s. Firstly, it opened the computing community to the masses and made it possible to construct a very powerful machine which could fit on a desk. No longer was it essential to log into the mainframe computer where CPU time and memory were limited and response was often slow - 'one per desk' became the motto of computer manufacturers. Secondly, the availability of small, reliable and above all relatively cheap microprocessors meant that the idea of multi-processor parallel computing could be given serious practical consideration.

2.1.2 The Need for High Performance Computers

There are numerous tasks which can be usefully handled by a computing machine but which are extremely computationally intensive and often require the entire computing power of a high speed von Neumann machine in order that the task can be completed in a reasonable time. Such tasks include various simulation problems, and tasks related to engineering design such as circuit placement and routing, and complex scientific and mathematical calculations (Hillis, 1986). Other tasks often cannot be carried out at the required rate even with an entire single processor machine. These include processing electrical signals in real time (Signal Processing) such as in radar, processing images, (for example the reconstruction of medical ultra sonic scanner signals), pattern recognition etc. It is tasks such as these which require special attention and the solution has been to introduce parallelism into computing. (Hockney and Jesshope, 1981).

2.1.3 Changing Attitudes to Computer Design

The need for a fundamental change in the design of special purpose computers away from essentially serial, von Neumann machines towards machines employing some form of parallelism is also necessitated by the same technology which makes parallelism a practical possibility. Since 1950, the speed of the components used in the construction of computers has increased by a factor of about 1000, from $1\mu s$ to less than $1ns$. Furthermore, the number of transistors per chip has approximately doubled every two years since the first integrated circuit was developed (Nomura 1985). This has been achieved by a combination of both the technological progression from valves to silicon transistors as described above, and of improved processing methods which have allowed devices of ever decreasing geometries to be reliably fabricated. Currently, devices at about the $1\mu m$ to $2\mu m$ gate length level are in commercial production with sub-micron devices having been sucessfully demonstrated at the research level. As geometries are scaled down further,

it will obviously be possible to integrate even more components onto a given area of silicon. However, a new problem rears its head in that although the propagation delay of devices decreases linearly (to first order) with reducing geometries, the delay associated with the interconnecting tracks remains approximately constant. It follows that system clock rates will become more and more dominated by the track delays as devices become smaller. As a result, it will be difficult to achieve *significant* increases in system throughput simply by producing devices with smaller geometries. This technological barrier to increased speed has been an important factor in the rapidly increasing interest in parallel processing. (Meindl 1985)

2.2 The Introduction of Parallelism

The earliest known reference to parallelism in a computing machine is thought to have been in 1842, by L F Menabrea (Kuck 1977), who referred to a machine capable of computing products of pairs of 20 digit numbers when used for repetitive tasks like the generation of numerical tables. Menabrea suggests that the computing machine could be arranged to provide several results simultaneously. Parallelism today means much more than just producing several independent results simultaneously of course, it includes the concept of partitioning a large problem into smaller sub-problems which although may be largely independent of each other must nevertheless cooperate to produce a combined output result. For this reason both the hardware and the algorithm should ideally be considered together and the process is aptly described by the term *Algorithmic Engineering.*

The transition from serialism to parallelism in computing has taken many decades to reach its current position even though structures for parallel machines had been proposed in the 1950s. The delay of about 30 years before any serious commercial machines appeared is largely due to limitations in technology which made arrays of processors very difficult and costly to implement. Until about the mid 1970's only architectures based on a single

stream of instructions and data had achieved any commercial success, notably the CRAY-1 in 1976. Of the many theories and structures based on multi-computer architectures proposed around 1950, the work of von Neumann was probably the most notable. Von Neumann carried out a theoretical study which showed that a 2-dimensional array of computing elements with 29 states could perform all operations because it could simulate the behaviour of a Turing machine (von Neumann, 1966). Unger (1958) proposed practical structures based on von Neumann's work. This early work on arrays of processors can be considered to have been the progenitor of the SOLOMON, ILLIAC IV and ICL DAP machines which appeared in the 1970's (Hockney and Jesshope, 1981). Similarly, Holland (1959) describes an assembly of processors each obeying their own instruction stream - an early vision of todays Transputer arrays. With the arrival of LSI/VLSI in the 1980's, small, reliable and cheap processors became available and permitted array architectures to be considered seriously. The first array computer was the SOLOMON computer (Simultaneous Operation Linked Ordinal MOdular Network) and is described in Slotnick et al (1962). This was a two-dimensional array of 32x32 processing elements each with a memory for 128 32-bit numbers and an arithmetic unit which worked in a bit-serial manner under control of a single instruction stream and control unit. Although never built as described in the 1962 paper, it led to the development of the ILLIAC IV, the Burroughs PEPE floating point processor arrays, the Goodyear Aerospace STARAN and the ICL DAP, all arrays of single-bit processing elements. (Hockney 1981)

2.2.1 Classification of Computer Architectures

Computer architectures can be classified according to their structure (Shore 1973) or according to the relationship between instructions and data (Flynn 1972). Although Flynn's classification is less precise than Shore's, it seems to be more popular and involves four classes as follows:

1. **SISD** - Single Instruction stream/Single Data stream. This is the conventional Von Neumann machine in which there is one stream of instructions executed by a single processor operating on a single stream of data. It is possible to imagine an SISD machine with more than one processor, but it is clear that such a machine does not provide any more computing power than the single processor version!

2. **SIMD** - Single Instruction stream/Multiple Data stream. This classification implies parallelism since it describes computers in which the same instruction is executed on several streams of data simultaneously. Vector machines such as the CRAY-1 are included in this category as well as processor arrays such as ILLIAC IV, ICL DAP and the Connection Machine from the Thinking Machines Corporation.

3. **MISD** - Multiple Instruction stream/Single Data stream. This class is essentially void since it implies that several instructions are being executed on a single data item simultaneously.

4. **MIMD** - Multiple Instruction stream/Multiple Data stream. This class implies parallelism and the ability of the machine to simultaneously execute several different instructions on different data. An example of a MIMD machine is an array of transputers (Inmos, 1984) each of which can potentially contain a different program. The PASM machine (Siegel et al, 1981) can also be considered in this category. PASM is essentially a SIMD array but is partitionable, with each partiton being controlled independently. It therefore has SIMD/MIMD capability.

Of the above categories it is the SIMD and MIMD architectures, both involving multiple processing elements (generally arrays) which are of interest in this thesis. In particular we are interested in two dimensional arrays of interconnected processing elements.

2.2.2 Pipelined Parallel Architectures

The programmable computer architecture is not the only application area for parallelism. There are many functions in the field of signal and image processing which benefit from both parallelism and pipelining of operations but where each processor performs a fixed task. A particularly important dedicated architecture of this type is the systolic array which was proposed by Kung (1980).

Systolic Arrays

A systolic array is an array of identical processing elements each of which communicates only with its nearest neighbours with all such communication taking place via latches. The latches are all clocked in synchronism. The function of the array is determined by the function of the processors and the way in which data streams are arranged to flow through the array. Systolic arrays have many important properties including total physical and electrical regularity, a high degree of parallelism through the use of many processors and high throughput rates due to inherent pipelining. The parallelism and pipelined nature of systolic arrays makes them ideal for use in signal processing applications where high throughput rates are essential to cope with real time data, but where the latency due to pipelining is not a serious problem.

Probably the best known systolic array is that proposed by Kung (1980), for the multiplication of banded matrices. This consists of a diamond shaped array of hexagonally interconnected processors each of which performs a multiply-accumulate function as illustrated in figure 2.1. Briefly the operation of the circuit is as follows. Data words enter the cells as shown and are multiplied together within the cell. The product is added to an incoming sum and the combined result passed out northwards. Each processing element therefore computes part of the final result and by the time a result emerges from the top of the array it has been fully formed.

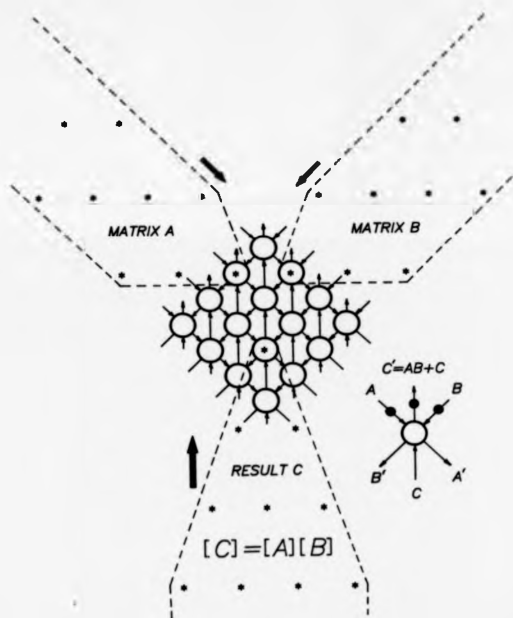


Figure 2.1: Systolic Matrix × Matrix Multiplier after Kung (1980).

Bit-level Systolic Arrays

Many of the systolic arrays proposed by Kung and others involved cells which performed a multiply-accumulate function. McCanny and McWhirter (1981) recognised that the multiply-accumulate function itself possessed inherent parallelism and showed how a systolic multiplier could be constructed in which each cell operates at the bit level. This was a significant step forward for both signal processing and VLSI design. For signal processing it meant that pipelining had been introduced at the bit level and that systems based on such arrays would have a very high throughput. For VLSI the implications are several. Firstly, because large numbers of bit-level cells can be integrated onto a single chip, entire functional blocks, for example multipliers, correlators and convolvers (Urquhart and Wood 1984, Evans et al 1983) can be implemented as single chip systolic systems. Secondly, since all cells in a systolic array have only nearest neighbour connections, the communication problems within the chip are simplified by avoiding the need to broadcast data. Furthermore, since all cells are physically and electrically identical, arrays can be scaled easily and so systolic arrays of arbitrary size can be designed without the need to worry about timing problems.

Wavefront Arrays

As we have seen, a systolic array involves data streams which move across the array in a completely synchronous fashion. The correct timing of the interactions between individual data elements is ensured by having each cell in the array controlled by the same clock. The systolic approach has advantages in terms of simple control and predetermined array timing, but does not necessarily result in a system with the highest possible performance. This is evident from the fact that the clock period must be sufficiently long to allow the slowest cell to complete its calculation. However, some cells may have completed their calculation well before the end of the clock period and could in principle start a new calculation without waiting for the next clock

period to start. Such cells are therefore essentially *held back* to keep them in synchronism with the slowest cells.

The wavefront array concept (Kung and Gal-Ezer, 1982) attempts to extract the fullest possible performance from a system. Instead of using a global clock to synchronise communications between cells, an asynchronous handshaking mechanism between cells is used. The handshaking procedure allows a cell to proceed with a calculation as soon as it has passed on the previous results to another cell and received the necessary inputs. This means that the data streams traversing the array will not have linear wavefronts, but will have regions where parts of the wavefront have become more advanced than the rest. In an array in which the cell computation time is a function of input data rather than spatial position, the average effect should be a net increase in system throughput.

2.3 The Need for Fault Tolerance

There are three main reasons why designers are interested in fault tolerance:

- To completely mask out the effect of faults occurring in service in safety-critical applications,
- To enable fast repair of faulty systems by manual initiation of a fault tolerant scheme,
- To enable fabrication faults in monolithic devices to be circumvented so that device yields can be increased and ultra large circuits can be built.

2.3.1 Reliability Improvement

The application of fault tolerant techniques to the above problems has evolved over many years because of the particular need at the time. The earliest need was for an improvement in the reliability of computers and is as old as the first computers, which were inherently unreliable due to the valves and relays

used in their construction. The move to solid state devices brought with it a dramatic improvement in reliability, which together with improved construction techniques such as printed circuit boards, largely alleviated the problem for all except the most complex systems.

As more and more complex systems continued to be built, much greater emphasis had to be placed on reliability, for example in the US space program of the early 1960s (Avizienis, 1976). An additional motivation was that increasing amounts of electronic circuitry were finding their way into other safety critical applications such as the control of aircraft and large factories. Human life depended on the correct operation of these circuits and much effort went into the study of techniques which could increase the probability of correct circuit operation over specified time intervals. (Siewiorek and Swarz, 1982).

In more recent times, the move towards parallel processing has enabled more complex machines to be designed and built. Even with highly reliable components which have survived the burn-in procedure, in-service failure can be a serious problem. However, although the hardware complexity which can readily be achieved in parallel processing systems brings with it problems of potential unreliability, the nature of parallel processing architectures also holds the key to their solution. The key lies in the regular structure of parallel processing machines, which are ideally suited to the inclusion of spare or redundant elements which can then be used to take the place of any which become faulty during operation. It is this feature which has led to much research on techniques for configuring processor arrays.

2.4 Wafer Scale Integration

The regular nature of processor arrays and the relative ease with which fault tolerance can be introduced into them has stimulated interest in whole wafer integrated circuits, or Wafer Scale Integration (WSI). Such circuits represent the ultimate goal of integrated circuit designers. However, wafer scale circuits

are impossible to fabricate successfully without some form of fault tolerance since the probability of at least one defect being present in the circuit is almost unity. In WSI the idea is to fully utilise the available silicon area on the wafer to fabricate a single monolithic device. In the case of a regular array of processing elements this avoids dicing the wafer into separate chips, testing, packaging and reassembling into an array similar in topology to the way the processors were arranged on the original wafer.

WSI is expected to be capable of providing higher performance systems due to the close proximity of the processing elements and the fact that it is not necessary for the elements to communicate via the high capacitance world which exists outside the monolithic silicon. The advantages of reductions in size, weight and power follow in a straightforward manner. However, unlike conventional systems which are constructed using selected components which have been found fully functional after thorough testing, WSI circuits have a fixed set of processing elements. Because the yield of the processing elements on the wafer is far from 100%, the ability of WSI circuits to tolerate fabrication defects is essential.

Faults in the elements on the wafer can be caused in many ways, but perhaps the most common are short and open circuits in the wiring, pinholes in the oxides, and dust particles affecting the etching processes. The larger the elements on the wafer, the greater the probability that any one of them will contain a fault. Extending this concept to a circuit covering most of the surface of a wafer, it becomes easy to see that a WSI circuit would never be fully functional without some in-built fault tolerance.

2.5 Types of Fault Tolerance

It must be stated at the outset that an essential requirement of any approach to fault tolerance is the inclusion of some form of redundancy. This can be in the form of either time or hardware.

2.5.1 Time Redundancy

Time redundancy implies that all the circuitry in the fault tolerant system is necessary if the system is to operate at its rated maximum speed, but that failures of individual elements of the system do not cause failure of the entire system even though the failed elements are not replaced. The system design is such that the remaining functional parts of the system perform the tasks which would normally have been carried out by the faulty elements. Since the total amount of hardware in the system is fixed, the speed of operation of the system is reduced. An analogy is a factory employing men to perform individual tasks which when combined result in the generation of a product. The full complement of staff is fixed and when all present the factory achieves its maximum product output rate. However, when one or more of the men is ill or on holiday (ie faulty), the product output of the factory will drop but probably not fall to zero since the tasks of the missing men will be shared by those present. The main problem of implementing time redundancy in hardware is designing an efficient algorithm for sharing the tasks among processors; one technique is presented by Sami (1984).

2.5.2 Hardware Redundancy

Unlike time redundancy, hardware redundancy allows the system to operate at its rated speed even when faults (up to some maximum number) are present in the system. Perhaps the most obvious form of hardware redundancy is the use of extra hardware elements and the inclusion of some arrangement for replacing a faulty part of the circuit with a spare. In other words to completely remove the faulty element from the circuitry being used. Techniques of this type are discussed in detail in Chapter 4 and will therefore not be pursued here.

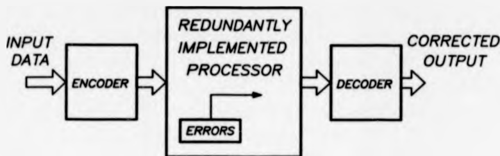


Figure 2.2: Typical arrangement for algorithmic fault-tolerance.

2.5.3 Algorithmic Fault Tolerance

It is also possible to design systems with hardware redundancy which have the redundancy built into the algorithm. A typical arrangement is illustrated in figure 2.2. In these systems, all of the hardware is used all of the time with redundant calculations taking place in the redundant hardware. Instead of inhibiting any faulty elements from taking part in the computation process by switching them out of the system, they are allowed to remain in place and generate faulty outputs. Data signals are encoded before entering the system and any faulty signals generated within the system propagate to the outputs. Here they are detected and corrected in a decoding process. This approach is called *Algorithmic Fault Tolerance* since faults which occur during the execution of the algorithm can be tolerated. The main advantage of algorithm fault tolerance is that it is able to cope with intermittent as well as permanent failures, completely masking the effect of the fault from the output without any loss of information. In a well designed system of this type the user need not be aware that a fault had occurred.

Systems based on algorithmic fault tolerance include transmission systems using error correcting codes, where each data word to be transmitted is encoded with a number of *check* bits. The check bits are subsequently used by a decoding circuit to correct errors in the received data word. A good treatment of transmission codes is given in MacWilliams and Sloane (1977). Error correcting codes have also been used in memory chips to detect

and correct both fabrication and in-service *soft* errors. (Cliff 1974). Other techniques are based on AN codes which involve pre-multiplying operands by a constant value A and checking that the result is a multiple of A. AN codes are useful for detecting errors which have occurred in arithmetic operations like addition and multiplication. A discussion of these and other coding techniques is presented by Wakerley (1978).

2.6 Scope of the Project

In this project we address the problem of applying *hardware* fault tolerance to 2-dimensional arrays of identical processors with nearest neighbour interconnections since such circuits are becoming increasingly important in many areas of signal and data processing. We will focus our attention on techniques suitable for use in the fabrication of large area integrated circuits since this is probably the most difficult problem of fault tolerance. However, the techniques can also be used in other applications.

In chapter 3 we further motivate the work by investigating the distributions of faults which occur in processed silicon wafers. Then in chapter 4 we review the state of the art in hardware fault tolerance techniques with examples of their application to both linear and 2-dimensional arrays. As we shall see, the work is particularly relevant to parallel processing, and due to the regularity of the arrays used in this field, efficient fault tolerant schemes can be developed.

Chapter 3

Fault Distributions in Integrated Circuits

3.1 Introduction

In chapter 2 we described the increasing move towards parallel circuit architectures, such as arrays of processing elements, for high performance computation. Typically, a single processing element might be integrated onto a chip and the chips then interconnected in the form of an array as required. The regularity of processor arrays naturally leads one to consider the possibility of integrating the entire array onto one large chip so that the tasks of dicing, packaging, re-testing, and assembling of the individual processors into the required array are eliminated. This approach is termed *large area integration* or more commonly *wafer scale integration* (WSI).

One of the problems of WSI is that the large area of the circuit means that the probability of at least one fault-causing defect being present in the circuit is almost unity, and the yield of the circuit will be zero. As we shall see in chapters 4 and 5, it is possible to incorporate redundant circuit elements so that faulty elements can be replaced. However, the effectiveness of redundancy depends strongly on the fault distribution (Mangir 1984). For this reason it is important to have an understanding of the distribution of the faults to be tolerated before attempting to incorporate redundancy into the circuit. In this chapter we discuss the main fault-producing mechanisms

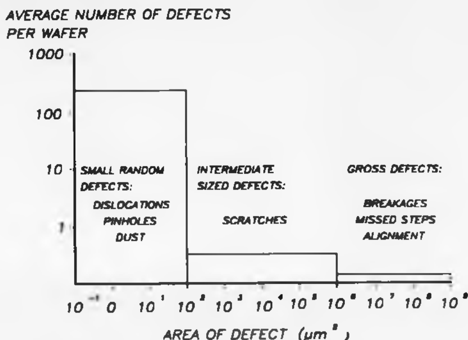


Figure 3.1: The main Classes of integrated Circuit Defect.

which occur in silicon integrated circuits and consider how they are distributed over the wafer. We then investigate the problem of modelling these distributions mathematically and present some of the models most commonly used in the literature. Finally we draw conclusions about the approach to fault tolerance which would be best suited to overcoming faults in a large integrated circuit.

3.2 Types of Defect in Integrated Circuits

Defects are present in wafers produced by any semiconductor process line. These defects can occur due to a variety of different causes and may be classified into three main groups according to Peltzer (1983), as shown in figure 3.1. The diagram is for a typical semiconductor wafer and shows the frequency of occurrence of each of the three defect types as a function of defect area. On the right hand side of the figure, we see that large area, or *gross* defects occur much less than once per wafer. These defects tend

to affect large portions of the wafer or even the entire wafer and render it unusable. Gross defects can be caused by breakages, missed processing steps, poor mask alignment, or incorrect processing, which could result in shifted device parameters. Many of these causes of defect can be detected during the processing, and further processing of the wafer can thus be avoided. Hovever, wafers with shifted device parameters may not be detected until after the processing stages have been completed.

At the other end of the scale of defect sizes are the *random* or *point* defects. A precise definition of when a defect is a point defect is difficult to provide, but defects which affect the operation of a single or small number of primitive devices such as transistors might typically be described as point defects. This means that defects with areas less that approximately $100\mu m^2$ would be considered to be point defects and occur in relatively large numbers, perhaps several hundred or so on a 4 inch wafer. Typical average defect densities for current processes are in the region of 2 to 5 defects per square centimetre (Chen et al, 1988). Point defects can be caused by a multitude of different mechanisms including dust particles, contaminated chemicals and pinholes in oxides, and will be discussed in more detail in the next section.

The class of intermediate-sized defects lies between the gross and point defects and includes defects due to residues remaining from photolithographic processes, small scratches, etc. They are largely superficial but nevertheless can severely detract from the overall chip yield if not properly controlled. Intermediate-sized defects occur typically as a result of poor handling and poor process cleanliness.

Stapper et al (1983) give the proportions of failures due to gross and random defects as shown in figure 3.2. As can be seen, typically over 80% of losses are caused by random defects.

Of the above categories it is the class of random or point defects which will concern us in the remainder of this chapter as it is the largest cause of yield loss in large area integrated circuits. In particular we will be interested in the *distribution* of the point defects over the surface of the wafer and how

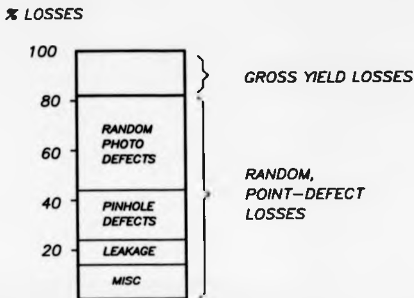


Figure 3.2: Typical Types and Proportions of Integrated Circuit Defect.

this influences the yield of chips on the wafer.

3.3 Random Point Defects

3.3.1 Causes of Point Defects

The Reliability Analysis Centre (RAC) in Rome collects data on all aspects of failure of many devices and systems. In table 3.1 we reproduce data collected by the RAC on LSI integrated circuit failures which occurred during initial testing after manufacture. This and other tables can be found in Siewiorek and Swarz (1982). The table shows that there are many types of point defect and that there is considerable variation between bipolar and MOS technologies. As one might expect, MOS circuits, being essentially surface devices, are more susceptible to defects in oxides and diffusions than are bipolar circuits.

The defects shown in table 3.1 are caused by imperfections in the processing, including the initial fabrication of the silicon wafer itself. They arise during processing due to random fluctuations in the conditions of the pro-

Failure Type	% Bipolar Failures	% MOS Failures
Surface	29	45
Oxide Defects	14	25
Diffusion Defects	1	10
Metallisation Defects	21	1
Interconnection Defects	29	4
Input Circuit Defect	1	8
Bond Defect	5	7

Table 3.1: Types of Integrated Circuit Defect.

cessing equipment and chemicals, and include fluctuations in grain size of metallisation, resistivities of polysilicon regions, small bubbles in solutions and resists (Lawson, 1966), quality of contact regions, step coverage of metallisation and contamination. These fluctuations are very difficult to control because they occur at such a microscopic level. Defects in the initial wafer preparation can include chemical inclusions and crystal imperfections which act as recombination or generation centres and can cause degraded device performance. (Mangir, 1984).

Size Distribution of Point Defects

Point defects occur in a range of sizes. A typical size distribution is shown in figure 3.3. This data has been gathered by Stapper et al (1983) and shows that the average defect size for this particular process is about $3.2\mu m$, but that the most frequently occurring defect size is between 1.5 and $2.0\mu m$. As one might expect, the measured frequency of the distribution decreases as the defect size increases. However, it also reduces for defect sizes below about $1.5\mu m$. This is due to the resolution limit of the photolithographic equipment; small defects on the masks are not sufficiently resolved to cause real defects to appear (Stapper et al, 1983).

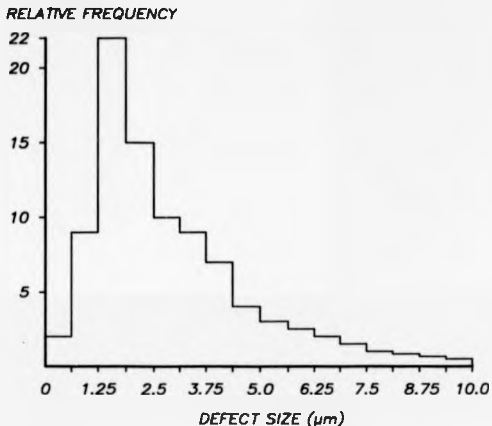


Figure 3.3: Typical Size Distribution of Integrated Circuit Random Point Defects.

Probability of a Defect Causing a Fault

The presence of a defect in an integrated circuit does not necessarily mean that it will cause a fault in that circuit. There are many areas within the circuit which are entirely blank (ie contain no components or interconnections) and these will be unaffected by the presence of a defect. In some cases defects occurring on components, especially interconnect, will be smaller than that component and may not cause a fault. An example of this is a small hole in a much wider metal track. The track will still operate in the presence of the defect. The problem of whether the circuit will experience reliability problems when in service due to electromigration in the region of higher current density around the hole in the track will not be considered here.

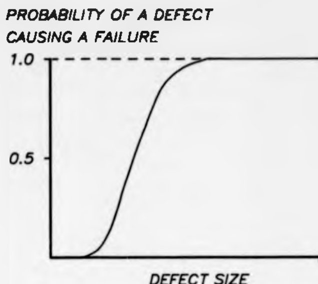


Figure 3.4: Typical Probability-of-Failure Curve for Random Point Defects.

It is possible to generate a curve of probability of failure as a function of defect size by using a defect monitor. A defect monitor is simply a special chip designed so that the number and effect of defects occurring on the chip can be readily and accurately measured. A typical probability-of-failure curve is shown in figure 3.4. As can be seen, very small defects typically cause no faults at all and have a zero probability-of-failure, while large defects always cause faults and have a value of unity.

3.3.2 Clustering of Point Defects

From the earliest days of analysing integrated circuit yield it was clear that the distribution of point defects was not entirely random over the surface of the wafer, and that defects tended to cluster together (Murphy, 1964). This tendency has been investigated by integrated circuit manufacturers who have employed inspectors to monitor the progress of wafers through the process line and count the number of particle defects on the wafer surface at each stage. This is done by shining a bright light obliquely across the wafer surface. The particles on the surface scatter the light and can thus be counted.

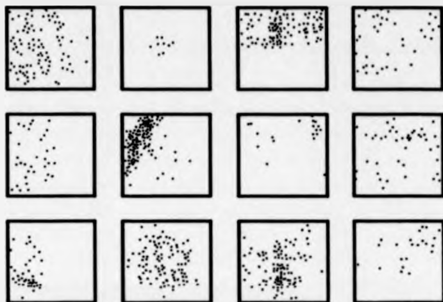


Figure 3.5: Typical Wafer Defect Maps showing Tendency to Cluster.

The wafers shown in figure 3.5 show results from a rather 'dirty' process line but clearly indicate the clustering tendency of the particles (Stapper, 1983). Stapper suggests that the clusters are caused by aggregates of particles which have collected in the manufacturing machinery and have been shaken loose by vibrations, pressures changes or gas flow changes. It is thought that these groups of particles will form clouds in gases and liquids used in the process line. Where the clouds reach the wafer surface, particles will be clustered.

Stapper also reports that edge clustering can occur while wafers are being held in 'boats', the carriers used to support wafers during processing. While in these boats, wafers can become contaminated from one side only and so will have more defects on the exposed edge than on the other parts of the periphery. Edge clustering has also been detected by Gupta and Lathrop (1972).

Clustering of defects is important in integrated circuits since it leads to higher chip yields than would be expected with purely random defect distributions. This is because the regions of clustered defects leave other regions relatively free of defects and these regions have a correspondingly

higher chip yield.

3.3.3 Radial Variations in Point Defect Distributions

It has been reported by several authors that the distribution of point defects on a wafer is dependent on the distance of the observed region from the centre of the wafer (Yanagawa 1972, Perloff et al 1981, Ferris-Prabhu et al 1987). The variations have been analysed by measuring the yield of devices across many wafers and plotting a wafer map containing the average yield obtained at each chip site. Their results are shown in figure 3.6 and it can be seen that all curves show distinct reductions in yield towards the edge of the wafer. It is also clear that a slight reduction in yield is experienced at the centre of the wafer. Explanations for these yield reductions have been proposed as follows. Perloff believes that the reduction in yield towards the edge of the wafer is primarily due to material defects, image distortion and mask registration problems. The yield reduction at the centre of the wafer has been associated with variations in thickness of the resist layers occurring during processing. During resist application, the wafer is first placed on a chuck and rotated at high speed. Liquid resist is then poured onto the centre of the wafer and the excess is spun off by the action of the rotational forces leaving a fairly uniform layer thickness over most of the wafer. However, the rotational force acting on the resist varies as a function of distance from the centre of the wafer, being close to zero at the centre. This results in a slightly thicker resist layer being present towards the centre of the wafer. Thicker resist layers will contain on average more defect due to their larger volume per unit area. In addition, in contact processes, where the photolithographic mask is placed in close contact with the wafer surface during exposure, more damage is likely to be caused to the wafer surface in areas of thicker resist.

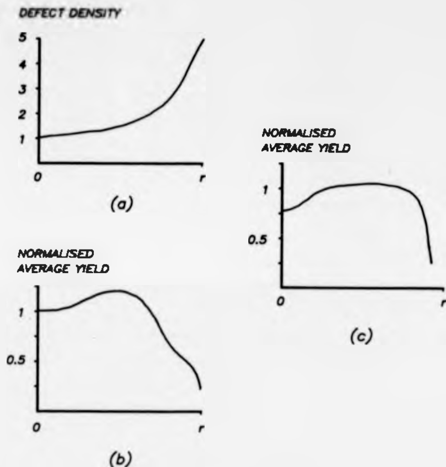


Figure 3.6: Variation of Chip yield and Defect Density with Radial Position on the Wafer. (a) Yanagawa (1972), (b) Ferris-Prabhu (1987), (c) Perloff et al (1981).

3.4 Modelling Integrated Circuit Yield

Ever since the development of the first integrated circuit, chip manufacturers have been interested in being able to model the process yield mathematically. Their interest in yield models is not merely academic but important for three main reasons:

1. **Process control**. Measurements from test chips included on the wafers passing through the process lines are stored in a database and used to evaluate the components of the yield model. These components

are plotted as a function of time and frequently reviewed and serve as an early warning of problems in the line. The problems can be rectified before serious yield losses occur.

2. **Product scaling.** When a product is being manufactured on a process line, it is possible to estimate the yield of another product if it were to be processed on the same line.

3. **Product planning.** When new products are being planned, yield models can assist in setting targets for future production.

In this thesis our interest in yield models is different. We require a knowledge of the relationship between chip yield and chip area so that a sensible choice of module area can be selected for use in a fault tolerant, large area integrated circuit. If too large a module area is chosen, insufficient functional modules will be available for configuration into the functional array. On the other hand, a choice of too small a chip area will result in a larger overhead of configuring circuitry per module.

Modelling integrated circuit yield is not a trivial task for two main reasons as follows.

1. Manufacturers are reluctant to divulge information about the yield of chips fabricated on their process lines because they believe that this would adversely affect their position in the highly competitive integrated circuit market. This means that most manufacturers have independently developed their own models for their processes. This results in many models with few common links.

2. To be accurate, a yield model must take account of the vast variations between wafers, batches and process lines as well as variations with time and with operator. This can result in very complex models containing many variables, some of which can be very difficult to determine.

As a result there are almost as many different yield models as there are researchers working in the area. Another problem seems to be that the

development of some models has been based on a very small sample of data and are not generally of use. It is a balance between model complexity and ease of use which is required.

Mathematical yield models have been proposed by many researchers. Price (1970) maintained that integrated circuit defects should follow Bose-Einstein statistics, while Gupta and Lathrop (1972) and Murphy (1971) thought that Maxwell-Boltzmann statistics should be used. Stapper (1983), on the other hand, investigates the use of generalised negative binomial statistics and shows that they are applicable to a wide range of chip sizes.

In the following subsections we present the simplest and most intuitive model based on Poisson statistics, a modified version of the Poisson model and finally a model which has gained wide acceptance, the generalised negative binomial model.

3.4.1 Poisson Distribution

The Poisson distribution is the simplest model of integrated circuit yield. It assumes that defects occur independently of each other at random positions across the wafer. The general form of the Poisson distribution applied to integrated circuits is

$$P(X = k) = \frac{e^{-\lambda}\lambda^k}{k!} \tag{3.1}$$

where $P(X = k)$ is the probability that k defects will occur per chip, and λ is the average number of faults per chip. The yield of the chips clearly occurs when $k = 0$, and is therefore given by

$$Y = P(X = 0) = e^{-\lambda} \tag{3.2}$$

λ is often written as

$$\lambda = AD \tag{3.3}$$

where A is the chip area and D is the average number of defects per unit area. The Poisson yield is illustrated graphically as a function of chip area in figure 3.7 for various values of D.

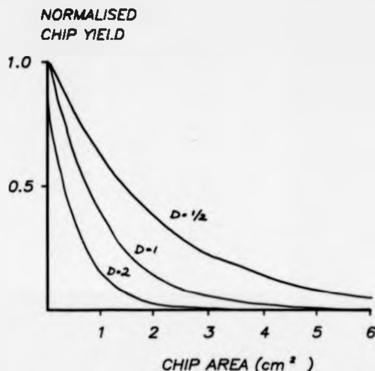


Figure 3.7: Chip Yield as a Function of Area as given by the Poisson Model.

It has been known for many years that the Poisson model is too simplistic to accurately represent the defect distributions found in integrated circuits. Stapper (1986) has shown this quite clearly using data from memory chips. Figure 3.8 shows the measured distribution of single-bit failures compiled from 450 memory chips fabricated on a modern process line. From the data, the average number of defects per chip is 28.6, while the percentage of chips with no faults, ie the yield, is 27.5%. Using the value of 28.6 for λ in equation 3.2, we obtain a yield of $e^{-28.6}$ or $3.8 \times 10^{-11}\%$; essentially zero! The discrepancy between the measured and calculated values for chip yield shows that the Poisson model does not represent the data.

It is clear from the data in the above illustration that the deviation from Poisson statistics is due to the clustering of defects. Clustering results in non-uniform defect densities with higher concentrations of defects in localised areas. This gives rise to other areas with relatively low defect densities, with

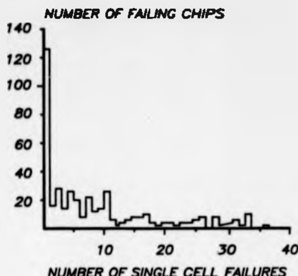


Figure 3.8: Typical Distribution of the number of Faults per Chip. (Average over 450 chips).

higher chip yields. For this reason clustering is actually beneficial in terms of increasing yield.

3.4.2 Compound Poisson Statistics

Compound Poisson statistics attempt to improve the basic Poisson model by making the average number of defects per chip a random variable. This enables the density of defects to vary over the surface of the wafer in a random manner. The wafer can be considered to be divided into a number of independent regions each having a random fault distribution, but each containing differing average numbers of faults. Each region is given an index number, i, and within each region, the Poisson distribution is assumed to be valid with an average number of defects equal to λ_i. Associated with each region i is a probability distribution, P_i. The compound Poisson distribution is then given by

$$P(X = k) = \sum_{i=1}^{\infty} P_i e^{-\lambda_i} (\lambda_i)^k / k! \tag{3.4}$$

resulting in

$$Y = P(X = 0) = \sum_{i=1}^{\infty} P_i e^{-\lambda_i}. \qquad (3.5)$$

The form of P_i is completely general and depends entirely on the nature of the fault distributions observed in manufactured chips. The problem arises in determining P_i since both the form and the parameters of the distribution must be matched to the process data.

3.4.3 Generalised Negative Binomial Statistics

Moore (1970) suggested the use of negative binomial statistics for modelling integrated circuit yield in the form

$$Y = Y_0(1 + AD/\alpha)^{-\alpha}, \alpha > 0 \qquad (3.6)$$

where Y_0 is a gross particle yield and α is a constant cluster parameter. Low values of α are associated with severe clustering, while as $\alpha \rightarrow \infty$, the model approaches the random defect distribution of Poisson statistics. For real wafers, α is typically in the range 0.5 to 4 (Ketchen, 1985). The yield model is illustrated in figure 3.9 as a function of chip area for various values of α. The values of Y_0, D and α must be determined from measurements made on wafers taken from the process line to be modelled.

Data of yield versus chip area can be determined for a process by a method using chip multiples which operates as follows. Wafers containing a large number of identical chips are processed on the production line to be modelled. Each chip is then tested and a map produced of the position of functional chips. From this the yield of the chips on each wafer can be calculated from N_f/N_t, where N_f and N_t are the number of functional chips and the total number of chips on the wafer respectively. The yield from the wafers can then be combined to give an average chip yield. The next step is to place a grid over each wafer in which each rectangle of the grid surrounds exactly two chips. The yield of the double-area chip can then be estimated by counting the number of rectangles in which both chips are functional.

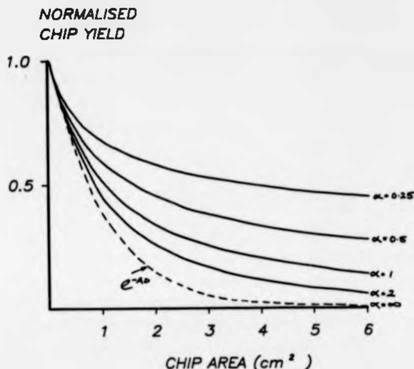


Figure 3.9: Chip yield as a function of chip area as given by the Generalised Negative Binomial Model.

This process can be repeated for larger chip multiples to produce a set of yield data for different chip sizes. The data can then be used to estimate the values of Y_0, D and α for the model by, for example, a non-linear least squares fitting procedure.

Stapper (1986) has used this procedure and shows that the model is representative over a chip area range of 1 to 36, the widest range ever published for which a single yield model is applicable. The model has also been found to give a good fit to data by several other researchers including Turley et al (1974) and Paz and Lawson (1977).

3.5 Implications for Fault Tolerant Techniques

In this chapter we have described the causes and effects of defects in integrated circuits and have presented some models which have been proposed

for estimating yield. Unfortunately, it is inevitable that much of what has been described has been general in nature due to the lack of real data available in the literature. To generate real data by carrying out tests on a process line would represent a separate thesis and is therefore not appropriate in the context of this work.

However, from the point of view of fault-tolerant integrated circuits, two points are important:

1. chip yield varies with radial position on the wafer, and
2. chip yield as a function of area can be determined using multiple chips for any given process. The ability of yield models to predict yields outside the range of available data is less important.

Furthermore, the wafer defect maps shown in figure 3.5 indicated that large areas of a wafer can often have a high density of defects. This makes the task of designing wafer scale circuits a difficult one since the clusters of faulty elements resulting from the defects seriously limit the ability to configure a functional circuit. For this reason, it seems likely that truly *full wafer* circuits will be limited to linear arrays and memory, for example, since such applications have few topological constraints. For two-dimensional arrays which are the subject of this thesis, it is more likely that large chips will be the main application, since these can make good use of the areas of lower defect density on the wafer.

Chapter 4

A Review of Hardware Fault-Tolerance for Processor Arrays

4.1 Introduction

In chapters 2 and 3 we have described the increasing interest in large integrated circuits, possibly up to the size of an entire wafer and have highlighted the problems of achieving this goal due to defects which inevitably occur in the silicon substrate or which are introduced during processing. It would be comforting to think that these defects could one day be eliminated. However, although standards of cleanliness during processing are being continually increased as a result of moving to smaller device geometries, it is unlikely that defect densities will reduce to zero for two main reasons. Firstly, the control of defects is a very difficult task, secondly, the use of reduced device geometries means that even if some of the larger defects can be controlled, smaller defects become more significant.

For these reasons, the ability to tolerate faults is essential if large area circuits are ever to be produced successfully. Fault tolerant techniques will be needed not only to overcome fabrication faults in highly complex monolithic circuits produced in the context of Wafer Scale Integration, but also to cope

The main body of this review chapter is to be published as a tutorial on Wafer Scale Integration as part of a book in Evans (1989a).

with in-service failures so that in the event of a failure, a system can resume correct operation after a short interruption, having been reconfigured around the detected fault.

In this chapter we classify and then review the techniques currently available for incorporating hardware fault tolerance into processor arrays, including the switch organisation and the method of implementing the switches. Many of the approaches which have been proposed in the scientific literature have not yet been demonstrated in real hardware. However, the details of some demonstration systems and devices have been published and these are included in the review wherever appropriate.

4.2 Classification of hardware fault tolerance schemes

Hardware fault tolerance techniques can be classified in two ways as follows:

1. according to the strategy for fault avoidance defined by the way in which the switching elements are organised,
2. according to the way in which switching elements are implemented.

Any particular fault tolerant scheme will be a member of a class from each category. The classes of switch organisation scheme are shown in figure 4.1 while the methods of switch implementation are shown classified in figure 4.2.

The switch implementation classification is essentially that of Katevenis and Blatt (1985) and is presented from left to right in order of increasing *lateness of binding*. This means that those techniques on the left hand side of the tree are fixed at the time of manufacture or configuration and are essentially permanent for the rest of the life of the device. Switch implementations further to the right become fixed progressively later in their life and have increasing facilities for re-configuration.

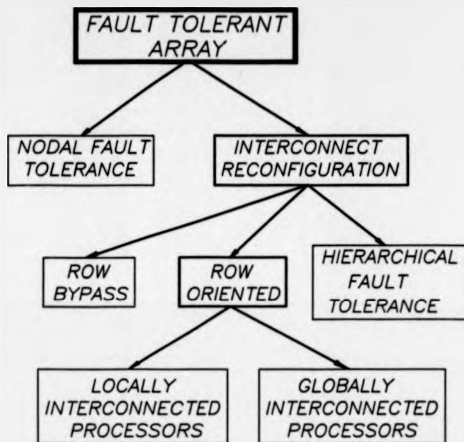


Figure 4.1: Classification of switching schemes for fault tolerant strategies in 2-dimensional processor arrays.

4.3 Switch Organisation and Configuration Schemes

In this section we briefly describe some of the approaches to configuration which fall into the classes shown in figure 4.1.

4.3.1 Nodal Fault Tolerance

The objective of Nodal fault tolerance is to increase the yield of the individual processing nodes within an array to 100% so that an acceptable overall array yield is achieved without having to configure the connection between nodes.

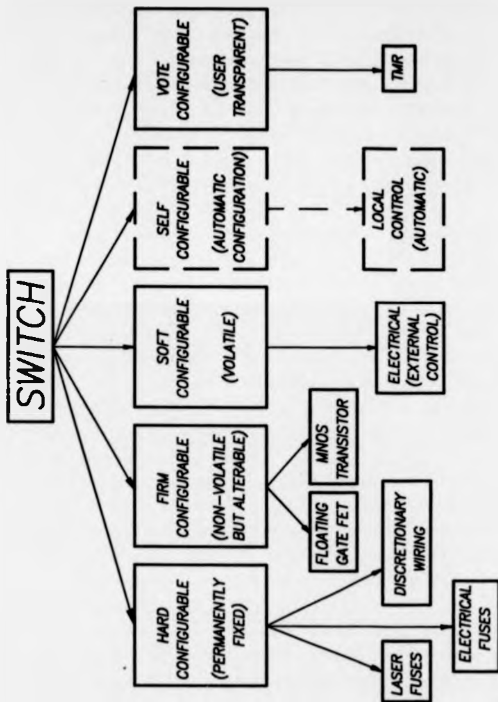


Figure 4.2: Classification of switch implementation methods used in fault tolerant integrated circuits.

The simplest method for doing this is to use Triple Modular Redundancy, or TMR for short. TMR involves using three processors in place of each of the original single processing nodes, together with a voting circuit. The voting circuit provides the output of the TMR node by delivering the majority verdict of the outputs of the three processors, all of which execute the same function on identical data. In this way any one of the processors can be faulty, but the voting circuit will still output the correct result. A TMR fault tolerant array is shown in figure 4.3(a) with an individual TMR node being shown in figure 4.3(b).

The TMR scheme has the advantage that it is very simple to implement since no configuration of the processors in the array is required at all, and no testing of the processors is required. It also has the advantage that it can tolerate in-service faults, both permanent and intermittent, without the user being affected or even needing to be aware that a fault exists. However, against these advantages there are some serious drawbacks. TMR systems require a large hardware overhead since each processor is now triplicated. Furthermore, the statistics involved in 2-out-of-3 majority voting schemes indicate that the yield of the processors involved must be quite high in order to gain any nodal yield advantage at all from using the scheme, and that even at best, the gain in yield is not dramatic. This can be seen from figure 4.4 which shows the processor yield and the TMR node yield. It can be seen that a processor yield of 50% is required before the node yield exceeds the processor yield. In addition it can be seen that the maximum gain in yield is achieved when the original processor yield is about 87%, at which point the node yield has risen to 95%.

The curve in figure 4.4 assumes that the voting circuits have a perfect yield, which would not be the case in practice. The effect of faults in the voting circuity can be reduced however by employing a voting circuit at the input to each of the triplicated processors. This scheme is illustrated in figure 4.5 and allows voting circuit faults to be tolerated since they now appear as faults on the input of a processor and will be treated as such by

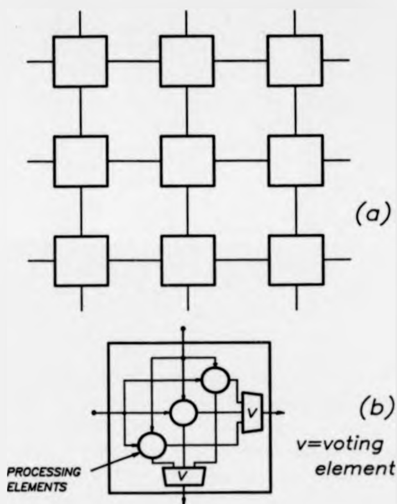


Figure 4.3: Technique of Triple Modular Redundancy: (a) TMR array, (b) TMR node.

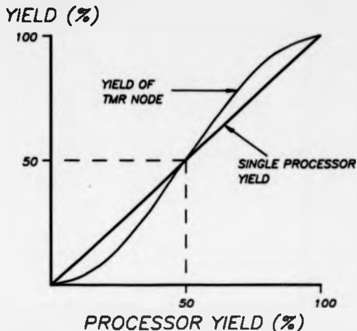


Figure 4.4: Nodal yield improvement achieved with TMR.

later voting circuits.

The very high hardware overhead of TMR for a small gain in yield has prompted the study of other methods of implementing nodal fault tolerance. One alternative method is to use two processors in place of the original processor, as against three for TMR. The idea is then to use a switch to select only one of the two processing elements for use in the array. This approach requires that the user knows which of the processing elements is working correctly and therefore implies that testing of the processing elements must be carried out. This could be done either by external test or by some form of self-test procedure. A technique using two processors per site was successfully used by Grinberg et al (1984) to increase the yield of individual wafers in their 3D computer based on stacked wafers. They used discretionary wiring to select between the two PEs on the node.

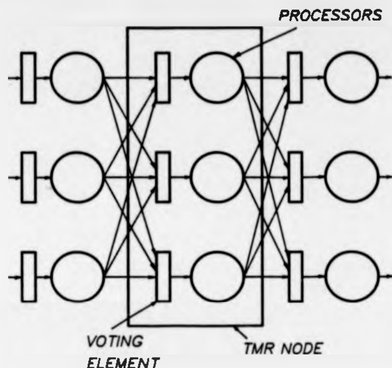


Figure 4.5: Technique for tolerating voting circuit faults in TMR.

4.3.2 Row or Column Replacement Schemes

As we have seen, nodal fault tolerance minimises or even eliminates the need to consider how to reconfigure an array to avoid faults and tries to sufficiently increase the yield of the nodes so that they can be connected directly into an array. Most hardware fault tolerance techniques however, rely upon some form of alteration to the connections between processors in order to generate a subset of the main array which is fully functional. In this way, faulty processors are completely isolated from the functional part of the array. The simplest technique of this type is the row-selection or row-bypass method.

Row Selection

The row selection technique is widely used in memory chips for yield enhancement (Moore, 1986), and is illustrated in figure 4.6. There are many

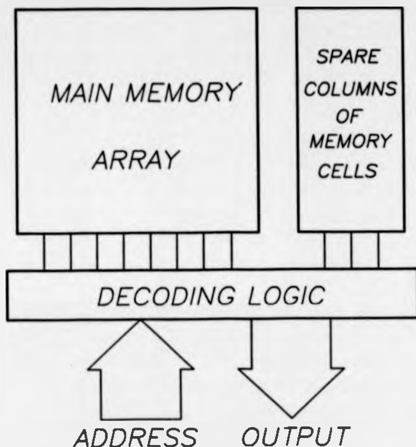


Figure 4.6: Schematic of a fault tolerant memory showing row de-selection technique.

ways in which the row selection procedure can be organised and several of these are discussed by Fitzgerald and Thoma (1980). The idea is to incorporate spare rows (or columns) into the array of memory elements and to use these rows to replace any rows from the main array which are found to contain faults. This technique is very simple to implement in memory chips because there are no signal interconnection paths between cells, and spare rows can be selected simply by programming the decoding circuitry appropriately. The decoder is often programmed by blowing electrical fuses.

Many memory manufacturers claim that the row selection technique is useful in the early stages of production of a device for increasing yield and

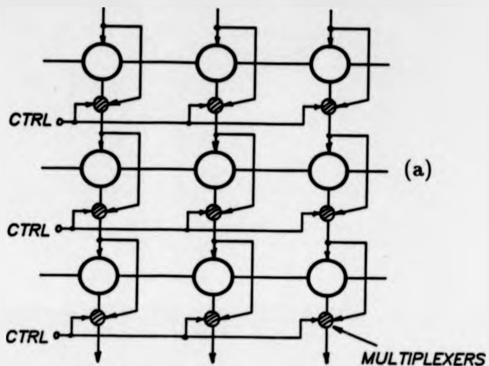
give figures ranging from 30 fold yield increase in immature processes, reducing to 1.5 fold yield increase in a mature process (Smith, 1981). However, NEC claim not to need fault tolerance at all; see Posa (1981), and Rogers (1982).

Row Bypass

In memory chips, rows are simply selected from those available so that sufficient cells are available for storing information. In processor arrays, a similar technique can be applied, but in addition, the connectivity between processing elements must be maintained. This means that when a row containing a fault is de-selected, its input signals must also be diverted to an alternative, functional row. This can be achieved most simply by employing bypass circuitry around each row so that the whole row can be bypassed in the event of it containing one or more faults. All spare rows are initially bypassed and the bypass is removed when the row is brought into operation.

A row bypass scheme like that described above was proposed by McCanny and McWhirter (1983), who also present figures which indicate that significant yield increases can be obtained. Figure 4.7(a) illustrates their approach and shows how multiplexers are used for the bypass mechanism. Figure 4.7(b) presents a graph of estimated yield increase which can be achieved, assuming a processing element complexity of about 10 gates. It can be seen that chips with a yield of around 10% without fault tolerance could yield at about 40% if fault tolerance were to be included. Similarly, chips with an initial yield of 1% might be increased to 20%. It is the latter of these two predictions which is likely to interest chip manufacturers since it could enable them to produce a larger device than previously possible and still retain an economic yield. This could enable the company to stay ahead of its competitors.

Moore et al (1986) extends the basic row bypass technique by considering the effect of an imperfect multiplexer yield on the overall array yield. He proposes various modified bypass circuits, some involving more than one



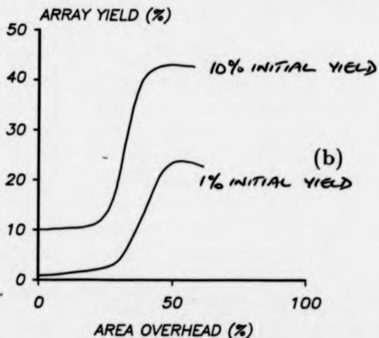


Figure 4.7: Row bypass scheme of McCanny and McWhirter (1983): (a) Array circuitry, (b) Estimated yield improvement.

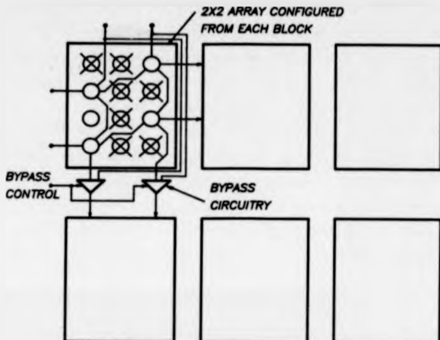


Figure 4.8: Hierarchical scheme of Hedlund and Snyder (1982).

multiplexer per cell, but which are thereby able to tolerate many of the faults which could occur in the bypass circuitry.

An important requirement of the bypass technique is that the yield of the individual processing elements is very high so that there is only a small number of faults in the array compared with the number of rows in the array. It is this constraint which enables the simple method of discarding entire rows to be beneficially employed. If too many faults are present in the array it is likely that many or even all of the rows will contain faults, and no increase in yield will be achieved.

4.3.3 Hierarchical Fault Tolerance

Hierarchical fault tolerance techniques have similarities to nodal fault tolerance. In the scheme proposed by Hedlund and Snyder (1982), illustrated in figure 4.8, processing elements are grouped into blocks of twelve out of which only four are required to work. The four working processors are then inter-

connected as a 2 by 2 subarray within each block and the blocks are then interconnected to form a 2-dimensional array. If any block in a row of blocks is unable to configure a 2 by 2 subarray, the whole row of blocks is bypassed. This scheme offers two levels of hierarchy and potentially allows a functional array to be configured from an array containing a large number of faulty devices, but also requires a very large overhead of redundant processors. It should be noted that although Hedlund and Snyder have chosen to bypass a whole row of blocks if a single block in that row cannot be configured into a 2 x 2 subarray, it would also be possible to use a more sophisticated strategy, such as one of those described in the next sections, for avoiding faulty blocks. In this way an improved yield characteristic might be achieved.

4.3.4 Row Generation Schemes

In these fault tolerant schemes, the idea is to generate functional rows of processing elements in which each functional row is constructed by taking one functional processor from each column of the array. The functional rows are then interconnected down the columns in the vertical direction to form the 2-dimensional array. Any faulty or unused processors encountered when interconnecting down a column are bypassed. Several row generation schemes are presented in the literature by Sami and Stefanelli (1983), Moore and Mahat (1985) and Bentley and Jesshope (1986). The schemes of Moore and Mahat are reproduced in figure 4.9 together with two columns of processors which have been configured by the techniques.

Scheme A is the simplest and operates as follows. Cell 2 can be connected to any one of cells 4,5 or 6. If cell 2 is to be connected to cell 5, switches F and G are opened, and E and H are closed. similarly, a connection between cells 2 and 4 requires that switches F,E,D and C be open and H,G,A and B be closed. One of the drawbacks of this simple scheme is that when a connection involving the use of the row-shift line is made, two other cells immediately become unusable, and one of these could be functional. To overcome this, scheme B employs extra switches and communication wiring to enable all

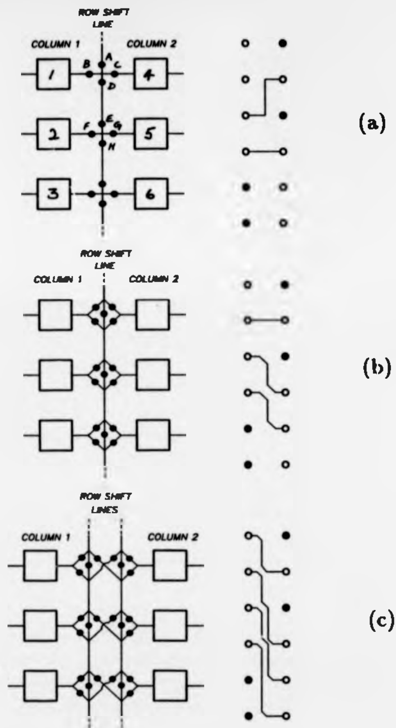


Figure 4.9: Row-oriented configuration scheme of Moore and Mahat (1985): (a) Scheme A, (b) Scheme B, (c) Scheme C.

cells to use the row-shift line simultaneously.

Scheme C is more sophisticated and allows double row shifts. This provides the cells with a greater degree of connectivity which offers increased flexibility to avoid faults.

The scheme of Sami and Stefanelli (1983) is similar to that of Moore and Mahat but allows as many row shifts as necessary. It therefore has superior performance but due to the high overhead is only suitable for array with a small number of spare rows.

4.3.5 Global Organisation

Schemes classified under the heading of global organisation offer a much greater flexibility as to how the cells are interconnected than other approaches. The most general scheme is probably that proposed by Katevenis and Blatt (1985) which is reproduced in figure 4.10.

The idea of global organisation is to provide the array with buses which run the entire length of the array between both the rows and columns. Switching nodes are inserted at the intersections of the buses so that connections can be selectively made between separate buses, and between buses and processing elements. In principle the global nature of the scheme means that a processor anywhere in the array could be connected to any other processor. This would provide an excellent ability for avoiding faults, but could lead to timing problems due to extra transmission delays being introduced into signal lines.

The operation of a global configuration scheme can be described generally as follows. First the buses are tested by an external tester, and then the working buses are used to give access to the switching nodes. These are tested and the combination of working buses and switching nodes is used to apply test patterns to the processing nodes. Finally the array is configured by setting the switches to the appropriate positions.

Many authors have proposed similar global configuration schemes. These include Hsia et al (1979), Raffel et al (1983) and Gaverick and Pierce (1985).

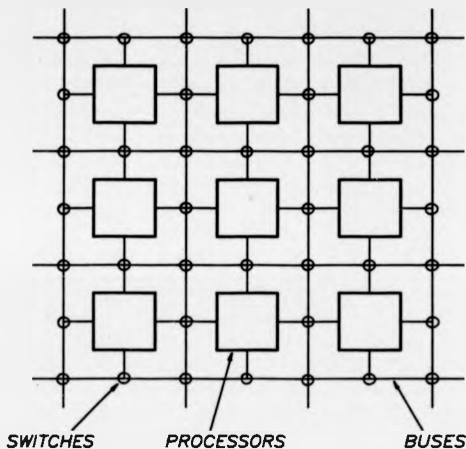


Figure 4.10: A global interconnection scheme after Katevenis and Blatt (1985).

The schemes differ mainly in the way in which the switching elements are implemented rather than having significant differences in configuration strategy.

4.4 Switch Implementations

In this section we review the methods by which the switches used in a configuring scheme can be implemented. The approaches have improved very significantly over the past two decades and can now provide a highly reliable interconnection medium.

4.4.1 Hard Configurable Schemes

The earliest proposals for fault tolerant circuits involved the use of hard configurable schemes, in particular discretionary wiring. Later proposals suggested using fuses at various parts of the circuit under the control of electrical heating or laser cutting. The laser cutting technique has been included in this section on hard configurable switching schemes since it has until recently been an irreversible process. However, recent reports indicate that the laser cutting processes may be reliably reversed, and this process will also be discussed.

Discretionary Wiring Approaches

The earliest attempts at increasing the area of integrated circuits were based on the principle of discretionary wiring. The idea is to place more circuit elements on the chip or wafer than actually required to perform the function and to test each of these elements by probing the wafer. The results of the test can then be represented as a wafer map and a metal mask can be designed which would interconnect the working devices. Sack (1964) proposes this approach for enabling whole wafer circuits to be produced. He demonstrates a complete wafer containing 108 gates interconnected in the form of a shift register. There are many variations on the basic discretionary wiring technique. Some are described in Petritz (1967), Lathrop (1967) and Calhoun (1969). All of these approaches appear to offer advantages at the level of integration available at the time (about 5000 gates on a 1 inch diameter wafer).

One of the problems with discretionary wiring is that it relies on there being very few faults in the wiring layer, which although achievable at the device geometries of the late 1960's, is unlikely to be successful at $1 \mu m$ geometries and with 4-6 inch diameter wafers. Another problem is that the probe testing of the devices on the wafer causes damage to the wafer surface which increases the probability of a fault occurring during subsequent

processing. An interesting approach along the theme of discretionary wiring is the approach proposed by Barsuhn (1978), in which he fabricates a wafer of memory chips. Faulty chips are replaced by good, individual mirror image chips which are flip-chip bonded over the faulty device. Barsuhn claims success with this method for a 2.25 inch diameter wafer.

Electrical Fuses

Electrical fuses are commonly employed as the method of implementing the necessary switching in yield enhancement techniques for memory chips, Moore (1986). They have also been extensively used in PROMs and Programmable Array Logic (PALs) for defining logic functions, although erasable techniques based on stored charge have now largely taken over. The electrical fuse technique is based upon heating the fuse, which is commonly made of aluminium or polysilicon so that it melts and creates an open circuit. When used in conjunction with pull-up or pull-down components, a change in logic level can be achieved and subsequently used to control other circuits. Although such fuses can in principle be combined in a circuit to allow a reversal of the effect of blowing a fuse by blowing a second, the fusing process itself is essentially irreversible. This means that the fusing technique cannot be used to isolate parts of a wafer for testing purposes and subsequently reconnect them.

Laser Linking and Cutting

The technique of using a laser beam to either cut or weld signal paths has been extensively studied at MIT Lincoln Laboratory by Raffel and his research team in the Restructurable VLSI (RVLSI) approach to large area integration (Raffel, 1983). The approach now seems to be a strong contender for Wafer Scale Integration due to its reliability and ease of execution. The structures used for linking and welding, together with details of the procedures used and some results are presented by Chapman (1985).

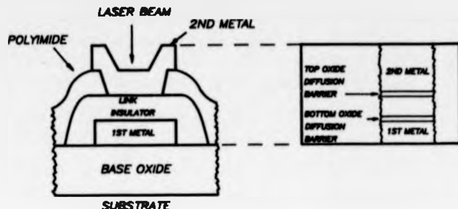


Figure 4.11: Cross-section of laser link from Chapman (1985).

Laser Power	$> 1.2\ W$
Pulse Width	$\approx 1\ ms$
Open Link Resistance	$> 10^{14}\ \Omega$
Formed Link Resistance	$< 1\ \Omega$
Failure Rate	$< 0.01\ \%$
Capacitance	$\approx 35\ fF$

Table 4.1: Laser link parameters.

The MIT stucture for making links between first and second metal layers is reproduced in figure 4.11 and details of the laser pulse required and the link parameters are given in table 4.1. During the linking process, for which an argon laser focussed to a $10\mu m$ spot size is used, successive melting of the second layer metal, the amorphous silicon insulator and part of the first layer metal occurs. This creates a silicon-aluminium alloy which provides the conducting path. An important feature of the melting process is that it occurs over a relatively long time period with a low power pulse ($1ms$ as against $100ns$ for commercial laser cutting systems). This avoids the splatter which normally accompanies metal vaporisation. MIT claim that the failure rate of the process is below that of the processing defects occurring during link fabrication.

The link structure described above also enables cuts to be made by using the laser to melt either the first or second layer metal just before it enters

the link structure. Cuts have been successfully carried out using the same low power as used for linking so that splatter is avoided.

4.4.2 Firm Configurable Switching Schemes

These schemes are characterised by switch reversibility combined with non-volatility of the switch setting. From the point of view of testing and configuring a wafer, this type of switching scheme is attractive, since areas of the wafer can be temporarily isolated while a detailed local test is carried out. Mistakes in configuration, or faults occurring after configuration can also be conveniently dealt with. There are two main approaches to Firm Configurable Switches, the Floating Gate FET and the MNOS transistor. These are descibed more fully in the following paragraphs.

Floating Gate FET

The operation of a floating gate FET switch (Shaver, 1984) is similar to a normal FET in that it is the voltage on the gate of the FET which determines whether the transistor is on or off. The difference is in the manner in which the gate voltage is applied. In a normal FET the gate voltage is controlled directly by applying a potential to a wire connected to the gate electrode. However, the gate of a floating gate FET is not connected to any source of potential but is determined by the amount of charge stored on the gate itself. This charge is deposited by irradiating the gate with a beam of electrons of the appropriate energy. *Normally-on* or *normally-off* FETs can be fabricated by selecting the appropriate channel polarity. Under irradiation by an electron beam, an n-channel depletion device is turned on, while a p-channel enhancement device is turned off.

An important feature of floating gate FETs is that the switch can be reversed by discharging the gate. This can be achieved in one of two ways; by standard ultra-violet irradiation or by electron beam irradiation. In the first of these techniques, the UV radiation allows the gate to discharge through photo-injection through the gate oxide. The UV radiation can be applied by

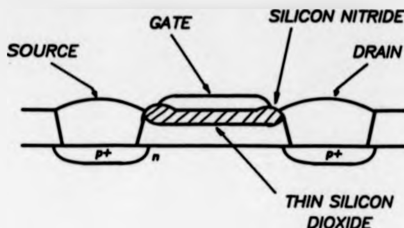


Figure 4.12: MNOS transistor.

a flood lamp or alternatively it can be localised so as to selectively discharge a single gate. The second technique, is attractive since it can be carried out in the same machine as originally used to charge the gate. A low energy beam is used which generates a secondary emission of electrons from the gate which is larger than the irradiating beam current. Since under these conditions, more electrons leave the gate than arrive at it, the charge on the gate reduces.

Although the floating gate FET switch is very attractive and will probably be acceptable for commercial devices, the retention time of charge on the gate may be too short for military devices (Shaver, 1984). However, the use of the floating gate FET in wafer scale integration is being investigated as part of the ESPRIT project number 824. An overview of this project is presented by Trilhe and Saucier (1987).

The MNOS Transistor Switch

The MNOS transistor illustrated in figure 4.12 is commonly used in Electrically Alterable Read Only Memories (EAROMS). These devices can store information for many years but can also be altered in a simple manner by the application of the appropriate programming signals which tend to be about 25 to 40 volts. The programming voltages cause injection of electrons into

the boundary region between the silicon nitride layer and the silicon dioxide layer. When the programming voltage is removed, the charge is retained since the boundary region is isolated. Erasure is achieved in a similar manner, with stored charge being repelled from the boundary and absorbed into the substrate. (see Muroga, 1982)

Although the MNOS transistor switch is simple in operation, it does have drawbacks in the context of wafer scale integration. The main problem is that a connection to control the programming would be required for each transistor and these would have to be accessible from the edge of the wafer. For small numbers of switches this may be feasible, but for large numbers, the problem will be serious.

4.4.3 Soft-Configurable Switching Schemes

The main type of soft-configurable switching scheme uses externally controlled electrical switching elements. This type of switch implementation is probably the one which most people would first think of. The idea is to design the switching nodes using ordinary logic gates. These are then controlled from an external source so that the desired connections are made between the processors. The great advantage of electrical switches is that they use only standard circuit components which are the same as those used for the remainder of the circuitry. In addition, since no specialised equipment is required, *re*-configuration can potentially be carried out in the field if in service faults occur. However, their main disadvantage is the same as for MNOS transistor switches, that the wiring needed to control them can become a serious problem. This type of switch implementation technique has, however, been used successfully by Anamartic (formerly Sinclair Research) in their wafer scale disk memory. Their switches are configured under the control of an external tester and the scheme is described in Aubusson and Catt (1978).

4.4.4 Vote-Configurable Switching Schemes

Circuits of this type have already been considered under the heading of nodal fault tolerance. However, although it is not immediately obvious, it is worth remembering that they are a form of electrical switch. Their advantage over externally controlled switches is that no wiring or global control is required; all the information they require to output the correct result is available locally. It is unfortunate that the hardware redundancy associated with the processing node to be used with this type of switch is so high as to be impractical in most cases, especially where the yield of the individual processors is low.

4.4.5 Self-Organising Switching Schemes

This type of switch is the subject of the remaining chapters of this thesis. The idea combines the convenience that external switches offer in terms of ease of implementation and potential for reconfiguration in the field but has the great advantage that no external control of the switches is needed. This not only removes the need for large numbers of extra pins just for configuration purposes, but also means that external computation to calculate the desired configuration pattern is not necessary; the ability to make decisions about the configuration pattern resides within the array cells themselves.

4.5 WSI Demonstrators

Much of the research which has been carried out on wafer scale integration has been limited to paper exercises backed up in many cases by computer simulations. There have been relatively few examples of actual devices being built although this is now changing and several demonstrations devices are currently being developed. In this section we describe some of the devices which have been fabricated and comment on those under current development.

4.5.1 Trilogy

Trilogy is probably the best known company involved in wafer scale integration. Their very ambitious project to build an IBM compatible mainframe computer on a single wafer received much attention in the press. The circuit was partitioned into about 1500 blocks containing between 10 and 50 gates each and triple modular redundancy was the method used to increase the block yield. In order to reduce the effect of clustered defects, the triplicated blocks were not placed adjacent to each other.

Unfortunately, Trilogy were unsuccessful in their attempt and investors have been wary of WSI ever since. Two main reasons have been suggested by Peltzer (1983) for Trilogy's failure. The first is that the spacing of the triplicated blocks led to an increased transmission time between blocks and as a result the project fell short of its target of an IBM compatible device due to lack of speed. The second reason is that the technology chosen for implementing the circuitry was ECL, and resulted in a power consumption of about 1kW on a 4 inch wafer. This led to tremendous problems of thermal management.

4.5.2 Anamartic and the Solid State Disk Memory

In the UK the wafer-scale memory built by Anamartic (formerly Sinclair Research) is probably the best known. The reason for this is that right from the start the device has been specifically aimed at the consumer market and as a result has received much attention in both the technical and national press. In addition, a novel configuration technique has been used and this has captured the attention of many observers. Working wafers with 0.5 megabits of storage were demonstrated in 1985. A higher density wafer is currently being developed as a commercial product which Anamartic hope will be able to replace conventional disks and have both much improved reliability and access time.

The technique employed by Anamartic for enabling the wafer to provide

sufficient yield is commonly called the *Catt Spiral* which was proposed by Aubusson and Catt (1978). The scheme generates a linear array of interconnected memory blocks starting from one block at the edge of the wafer and adding extra blocks to the chain one by one. The configuration is implemented using electrical switches which are controlled by an external tester/controller.

The implementation procedure operates as follows. The controller initially tests one of the chips on the periphery of the array. If the chip is faulty, another peripheral chip is chosen until a functional device is found. Instructions are then sent to the functional chip to tell it to connect itself to one of its neighbouring chips. The way in which the neighbour is chosen is described later. The chosen neighbour is then tested by the external controller by sending test patterns through the first chip, and into the neighbour. Test results are passed along the reverse route. If the neighbour is faulty, an alternative neighbour is chosen until a functional neighbour is found. This chip is then added to the chain. The chain is then further built up by repeating the process. If at any point in the configuration procedure a chip at the end of the chain is found to have no functional neighbours, it is removed from the chain and the previous chip in the chain is used as the new chain end. This backtracking ability can also enable the chain to escape from dead ends which may exist on the array.

The order in which neighbours are selected as candidates for the next position in the chain determines the shape of the final chain. In the Anamartic design, the most right-hand neighbour is selected and this results in a chain of good chips which hugs the outer edge of the array and spirals in towards the centre of the wafer. Figure 4.13 shows a wafer which has been configured in this way. For wafer scale integration it has the advantage that all the interconnects between elements of the chain are checked at the time of testing and as a result a chip cannot be included in the chain unless all wires are intact. Another advantage of the way in which the linear array is built up block by block is that the external switch control signals are applied

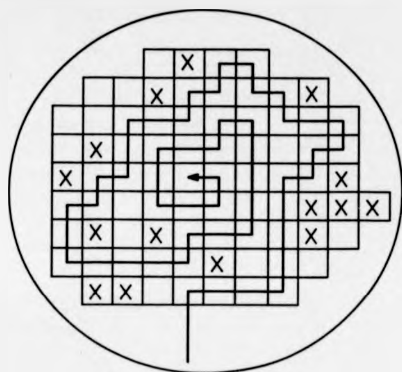


Figure 4.13: Wafer configured as a linear processor array using the spiral technique of Aubusson and Catt (1978).

serially. This avoids the need for large numbers of pins on the wafer.

Working prototype devices have been successfully fabricated and have demonstrated that the yield of the control circuitry on the cells is adequate, with around 90% of cells having working circuitry. As far as is known, no details of performance have been published, but in a public demonstration of a device, it was clear that a large proportion of the memory elements were also functional and could be connected to the spiral chain.

The use of electrical switches means of course that the spiral pattern generated by the test and configuration procedure is volatile and will need to be reapplied each time the device is powered up. The device could be retested each time, but Anamartic have chosen to store the configuration pattern in a ROM. The contents of the ROM can then be loaded into the wafer before it is used.

4.5.3 MIT and Lincoln Laboratory

The work at Lincoln Labs on wafer scale integration is based on their technique of Restructurable Very Large Scale Integration or RVLSI for short. Several demonstrators have been built successfully and a review of the progress of the project is presented in Rhodes (1986).

One of the demonstrators based on the RVLSI approach is a digital integrator consisting of 256 10-bit counters partitioned into 64 cells. Each cell contains four 10-bit counters. The complete device contains 130,000 transistors on a 4 inch wafer using $5\mu m$ CMOS technology. The configuration switches use 1,900 laser anti-fuses and 137 laser fuses.

4.5.4 GTE Laboratory

GTE are implementing a pipelined processor in WSI (Cole, 1985). The pipelined element contains a high speed sequencer, a micro-code RAM, a 32 bit ALU, and status and storage registers. Each element contains 150,000 transistors and 60 elements can be implemented on a 3 inch wafer. A self-test procedure is incorporated in each element. This checks the element itself and also the interconnections to the neighbouring elements. If a fault is found, an electron beam programmable switch is used to disconnect the offending processing element.

4.6 Conclusions

In this necessarily brief review of published techniques for incorporating hardware fault tolerance into processor arrays we have seen that the main difference between the techniques reviewed is in the method of organising the switches to obtain a better utilisation or *harvest* of the functional processors and in the way in which the switches are actually implemented. A common feature of many of the approaches is that the array is configured before being used in a system and is then essentially fixed for the rest of its operational life. In principle, the schemes implemented using electrical

switching elements could be reconfigured, but would require to be removed from the system before reconfiguration could take place. This is because the necessary test equipment for fault location, and the means for calculating the configuration pattern are separate from the array and are unlikely to be provided within the system.

For this reason there appears to be a niche for a fundamentally new approach to the problem of WSI and fault tolerance in general in which external control of the switches is not required and where configuration can be carried out by the array itself. Such a scheme would have obvious labour saving benefits in a WSI circuit but could also be extremely useful in other applications involving processor arrays including silicon hybrid circuits, and systems constructed from pcbs.

In the next chapter, we consider novel algorithms by which a two dimensional processor array containing faulty processors can *organise itself* into a functional array without any external assistance. In later chapters, the basic self-organising algorithms are developed into practical systems.

Chapter 5

Self-Organising Algorithms for Two-Dimensional Processor Arrays

5.1 Introduction

In Chapter 4 we have seen that many of the approaches to hardware fault tolerance in 2-dimensional arrays involve the use of electrical switching networks to allow faulty processors to be replaced by spares. In all cases the switches are set by some external controller which decides which switches should be used and how the array should be configured. The aim of this project has been to investigate techniques which enable an array to automatically organise itself around the faulty elements and generate a functional 2-dimensional array from an array containing many faults. Ideally the array would be able to do this without any external assistance. This would avoid the need for an external controller and would also provide a system which could readily be reconfigured if a fault occurs in service.

We have developed what we believe to be a novel solution to the 2-dimensional array configuration problem. In our technique, which we call *WINNER*, an acronym for *Wafer Integration by Nearest Neighbour Electrical Reconfiguration*, (Evans, 1985), each cell within the array is provided with some intelligence in the form of a small amount of additional control cir-

The main body of this chapter is to be published in book form in Evans (1989b).

cuitry. This is sufficient to enable each processing element to independently and simultaneously make local decisions about how it should be connected to neighbouring elements taking into account its own functionality, the functionality of its neighbours and the connection priorities to these neighbours which are defined in specific algorithms. The effects of these local decisions propagate throughout the array and manifest themselves globally as a complete self-organisation of the functional processing elements into a correctly interconnected functional 2-dimensional processor array. We call such arrays *Self-Organising Arrays.*

A number of algorithms incorporating the concepts of self-organisation can be derived. For the purposes of this thesis we concentrate our attention on two related algorithms which illustrate the technique. The first scheme applies the *WINNER* algorithm in one dimension of the array only, and generates functional rows of processors. A simple fault bypassing technique is then used to form the second dimension, (Evans, 1985). It is the simpler of the two algorithms and is presented in section 5.3. The second method applies the *WINNER* algorithm in two dimensions and is discussed in section 5.4, (Evans et al, 1985).

We describe the algorithms at a high logical and operational level. Detailed circuit level descriptions and results of performance simulations etc are presented in later chapters.

5.2 Definitions

The following terms will be used in subsequent description of the algorithms.

Processor: This is the circuit which performs the node function when the array is operating in-service. It communicates with four neighbours to the North, South, East and West.

Control circuit: This comprises the extra logic which is added to the processor to provide it with the self-organisational ability.

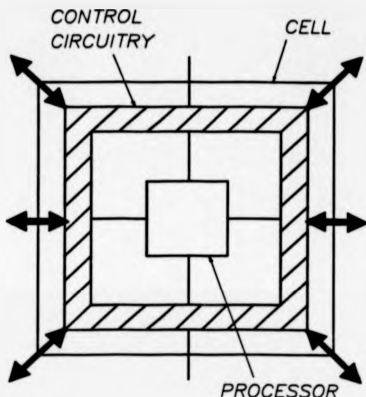


Figure 5.1: Relationship between Cell, Control circuitry and Processor.

Cell: A cell is the combination of the processor and its control circuitry and is illustrated in figure 5.1.

The assumption that the processor has only orthogonal connections does not limit the generality of the approach since all other nearest neighbour array interconnection structures can be reduced to the orthogonal form. An example of a hexagonally interconnected array and its orthogonal equivalent are shown in figures 5.2(a) and (b) respectively. The transformation has been carried out in the following manner. Firstly, the connections which are already orthogonal remain unchanged. Next, each diagonal connection directly connecting a cell to its south-eastern neighbour is re-routed via the eastern neighbour. This means that a dummy connection is required in each cell as shown in figure 5.2(d) instead of the diagonal connections shown in figure 5.2(c).

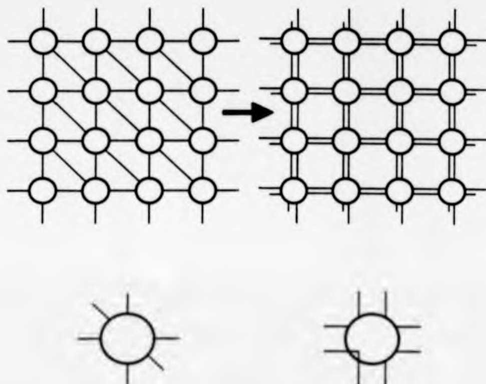


Figure 5.2: Hexagonally interconnected array and its orthogonal equivalent.

5.3 Algorithm 1: *WINNER* in One Dimension

The *WINNER* algorithm in one dimension is so called because it results in an array in which a number of functional *rows* have been generated. These rows avoid faulty cells but do not themselves form a two-dimensional array. The second dimension of the array is formed by interconnecting the functional rows in the vertical direction, bypassing any faulty cells encountered in the process. In order to describe the algorithm, we present an array which has been configured using the algorithm and explain how the configuration has been achieved by giving a simple pencil and paper description. We then show how this procedure can be implemented as a circuit and describe the operation in detail.

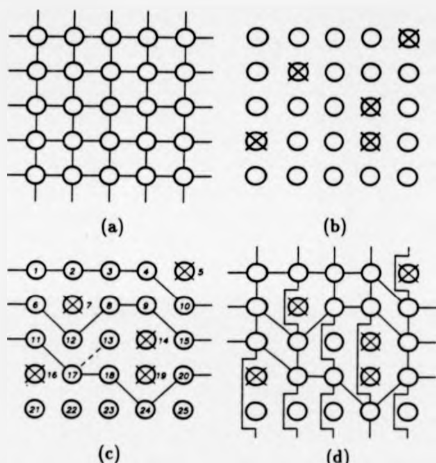


Figure 5.3: Configuration of a 2-dimensional array using *WINNER*: (a) Configuration of a perfect array, (b) Array containing faults, (c) Rows configured around the faults, (d) Vertical connections made: array configuration complete.

Figure 5.3 shows the main stages of an array which is being configured by the one-dimensional *WINNER* algorithm. Figure 5.3(a) shows how an array with no faulty processors would be configured. Figure 5.3(b) shows an array containing faults which is to be configured in the example, with faulty processors being indicated by a cross. On paper the functional rows can be generated as follows, with the cell numbers under consideration being shown in figure 5.3(c) and being referred to in brackets. Starting with the left hand column of the array, choose the functional cell nearest to the top of the array

(1). From this cell, look at the column to the right and make a connection to one of the three (in general) nearest neighbour cells in that column, (in this case 2 or 7, since the upper boundary of the array limits the choice to two cells). In making the choice always choose a functional cell, with a preference for the cell nearest the top of the array. Repeat this procedure until the right hand side of the array is reached, (2,3,4,10). At this point, one complete functional row (1,2,3,4,10) has been generated. Subsequent rows are formed in a similar manner treating the cells used in a previous functional row as if they were faulty cells (ie avoid using them). The second row would therefore be (6,12,8,9,15). The procedure minimises the distance of the rows from the top of the array and therefore maximises the number of rows which can be generated on the array.

In some rows a dead end may be encountered. This is illustrated in the generation of the third functional row in figure 5.3(c) and occurs when processor 17 connects to 13. None of the neighbours of processor 13 are AVAILable and so the row must backtrack to 17 and try 18. The row can then be continued to completion. In a large array several back trackings may be required on some rows. A fourth row in this diagram cannot be constructed because a complete dead end is encountered. The row must therefore backtrack to the boundary of the array.

Each functional row formed in this way contains a processor taken from each column of the array. The second dimension of the required array can therefore be generated by making vertical connections between processors of each column and bypassing faulty processors and processors which although not faulty are neverthless unused. This process completes the construction of the 2-dimensional array and is shown in figure 5.3(d). It is clear that the resulting array will be smaller than the original array due to the presence of the faulty elements but it should be noted that the x dimension of the functional array is identical to that of the given array, while the y dimension depends on the number of faults which occur in each column of the array.

5.3.1 Self Organisation

We now consider how this pencil and paper procedure can be embodied within a circuit so that it can operate automatically. To simplify the problem we make the following assumptions, the validity of which will be discussed in chapters 8 and 9.

1. Each processor contains some method enabling it to indicate reliably whether or not it is working. This could in principle be achieved by a self test procedure.
2. All connections between processors are fault-free,
3. Each control circuit associated with a processor is fault free,
4. All connections between control circuits are fault-free.
5. All data signals flow from left to right and top to bottom. This simplifies the description of the algorithms but in no way makes them less general.

In order to enable faulty cells to be avoided and to allow control circuits in adjacent cells to communicate, a cell with greater connectivity than the original processor is required. A schematic view of a cell with sufficient connectivity to perform the one-dimensional self-organising function is shown in figure 5.4. We can see that although the North-to-South (N-S) connection is unchanged, extra channels have been provided on the Eastern and Western sides for both inter-processor and inter-control circuit communication. These connections allow the cell to communicate with its nearest neighbours to the NW, W and SW, and the NE, E and SE directions respectively. Control circuits in neighbouring cells communicate via single-bit control lines indicated in figure 5.4 as *REQuest* (REQ) and *AVAILability* (AVAIL) signals.

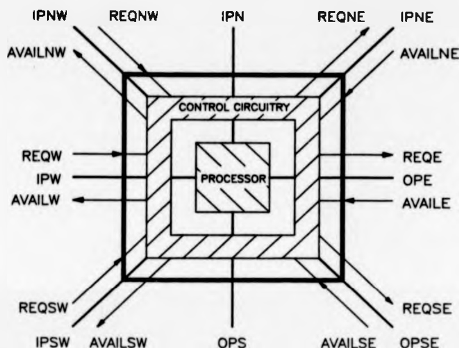


Figure 5.4: Schematic of cell suitable for 1-dimensional *WINNER* algorithm.

5.3.2 Control Circuitry

The function of the control circuitry in each cell is to decide how the cell should be connected to its neighbours in adjacent columns. In addition it must decide whether or not it should act as a bypass in the North-South direction. These decisions must be made on the basis of the following information, which is the only information available to the control circuitry:

- a knowledge of whether the processor in its cell is functional or faulty,
- REQuest inputs from its North-West, West and South-West neighbours, .
- AVAILability inputs from its North-East, East and South-East neighbours.

Using this information the control circuitry must perform the following functions:

- generate REQuest and AVAILability signals and route them to neighbouring cells,
- select data inputs from the appropriate neighbour and apply them to the processor,
- bypass the processor in North-South direction if the processor is faulty or unused.

Convention for REQuest and AVAILability signals

The following convention for REQuest and AVAILability signals will be used throughout this thesis:

> REQuest and AVAILabilty signals are *active* when at a logic 1 level and *passive* when at a logic 0 level.

Based on this convention, the phrases, *to send an AVAILabilty signal* and, *to send a REQuest* imply that the signals being sent are TRUE.

If a cell A outputs an AVAILability signal to another cell B it means that cell A contains a processor which is AVAILable for connection if REQuested by cell B. If a cell A outputs a REQuest signal to some other cell B it means that cell A wishes to set up a communication channel between its processor and the processor in B. Cell A can only send a REQuest to cell B if B is sending an AVAILability signal to cell A. If such a communication channel becomes set up then A is said to have been *connected* to B and the incoming data signals to cell B from cell A are directed to the processor in cell B.

The manner in which the REQuest and AVAILability signal are generated by a cell forms the heart of the algorithm and is described in detail in the following sections.

Generation of AVAILability Signals

A cell generates AVAILability output signals according to the following rules:

REQuest Inputs			AVAILability Outputs		
REQNW	REQW	REQSW	AVAILNW	AVAILW	AVAILSW
TRUE	X	X	TRUE	FALSE	FALSE
FALSE	TRUE	X	TRUE	TRUE	FALSE
FALSE	FALSE	TRUE	TRUE	TRUE	TRUE
FALSE	FALSE	FALSE	TRUE	TRUE	TRUE

X = TRUE or FALSE

Table 5.1: Generation of AVAILability output signals.

1. A cell can only output a TRUE AVAILability signal if it contains a processor which is fault-free (ie the self-test shows it to be functional), and at least one TRUE AVAILability signal is being received from its NE, E or SE neighbours.

2. If (1) is satisfied, then the priority system given in table 5.1 operates to decide in which directions to send TRUE availability signals depending upon the incoming REQuest signals.

From the bottom line of table 5.1 we can see that a cell receiving no REQuests will output a TRUE AVAILability signal to each of its three left hand neighbours. This allows any of the neighbours to make a REQuest if it wishes to do so at a later time. From the first three lines of the table we see that once a REQuest has been received, the cell may or may not remain AVAILable to the other neighbours depending on the priority of the REQuest. A REQuest from the NW neighbour has the highest priority, with the W and SW neighbours having successively lower priorities.

The scheme allows a priority of connections to be established so that a REQuest from the NW has highest priority, and REQuests from the W and SW have successively lower priorities. Such a scheme is required to ensure that a stable solution is reached. The scheme causes cells to output FALSE AVAILability signals to neighbouring cells if they have no chance of obtaining a connection. This occurs for example when a higher priority connection has already been established.

Rule (1) above gives the cell a global look-ahead capability even though each cell is capable only of local communication. This enables clustered faults to be avoided in the following way. Information is passed between cells from East to West about the availability of other cells. This allows a cell A to prohibit another cell from connecting to it if A either contains a faulty processor or would be part of a dead-end route, ie a route that would not be able to be completed due to some blockage later. Such a dead end route could occur, for example, if three vertically adjacent processors were faulty. In this case a functional processor to the left of the centre faulty processor would find that all of its possible connections to neighbours are unavailable. The functional processor would then declare itself to be unavailable. The scheme allows information about blockages to be transmitted from right to left to all the relevant processors, which then decide upon some appropriate avoiding action. These features will be illustrated in section 5.3.5

Generation of REQuest Signals

Request signals are output from a cell according to a different set of rules:

1. A cell can only output a TRUE request signal if its processor is fault-free and at least one request has been received from one of its NW, W or SW neighbours.

2. If (1) is satisfied then the cell outputs a single TRUE request value to one of its NE, E and SE neighbours depending upon the incoming AVAILability signals according to the priority given in table 5.2

These rules ensure that only one REQuest signal is output from any cell, which in turn ensures that a cell can never accidentally become connected to more than one neighbouring cell in any column.

5.3.3 Array Boundary Conditions

When a number of *WINNER* cells are interconnected to form an array, the inputs around the edges of the array are not connected to other cells and must

AVAILability Inputs			REQuest Outputs		
AVAILNE	AVAILE	AVAILSE	REQNE	REQE	REQSE
TRUE	X	X	TRUE	FALSE	FALSE
FALSE	TRUE	X	FALSE	TRUE	FALSE
FALSE	FALSE	TRUE	FALSE	FALSE	TRUE
FALSE	FALSE	FALSE	FALSE	FALSE	FALSE

X = TRUE or FALSE

Table 5.2: Generation of REQuest output signals.

be defined explicitly. The values of the boundary REQuest and AVAILability lines are defined as follows.

Boundary AVAILability Inputs

- Set NE and SE boundary AVAILability inputs to FALSE,
- Set E boundary AVAILabilty inputs to TRUE.

Boundary REQuest Inputs

- Set NW and SW boundary REQuest inputs to FALSE,
- Set each W boundary REQuest input to TRUE if corresponding W AVAILability output from the array is TRUE; otherwise set to FALSE.

5.3.4 Interaction of REQuest and AVAILability Signals

The availability and request signals together provide the cells with all the information they need about their surroundings in order to be able to form functional rows of interconnected processors. The priority system for sending and receiving request and availability signals ensures that stable functional rows are established from west to east and from north to south starting in the top left hand corner af the array. The priority system also ensures that each row formed is as close as possible to the northern edge of the array, thus maximising the number of rows generated.

The one dimensional *WINNER* algorithm operates automatically when the array is switched on and is a totally asynchronous technique. All cells continuously make decisions based on the information available to them. This information will be changing as the organisation of the array gradually evolves to its stable state. Therefore, in the early stages of self organisation the array may be highly dynamic with cells forming and relinquishing connections to other cells as a result of being overridden by higher priority decisions which have been made at other localities and have rippled through the array. Connections may in fact experience a number of iterations of this type but will always settle into a self-consistent, stable state.

The array can be visualised as having two levels of hierarchy. One level comprises the underlying asynchronous network of control circuitry which is capable of establishing communication channels between appropriate neighbours to generate a functionally orthogonal array as described. The second level is the array of processors containing a number of faulty elements which can be considered to be overlaid on the array of control circuitry which then forms connections between processors as appropriate.

5.3.5 Serial Description of *WINNER* Operation

Although the interactions of REQuest and AVAILability signals between the cells of the array occur simultaneously, it is helpful from the point of view of understanding the operation of the algorithm to consider the process as a *sequence* of distinct events as follows.

We assume that initially no REQuest or AVAILability signals are present in the array other than fixed boundary input values. From this starting point, no REQuest signals can be generated by any cells until at least one AVAILability signal has reached the left hand side of the array; this is the first stage of the configuration process. The AVAILability signals from each cell to its neighbours are generated starting from the right hand column of the array and working column by column across the array. For each column, the three availability outputs of each cell in the column are set to TRUE

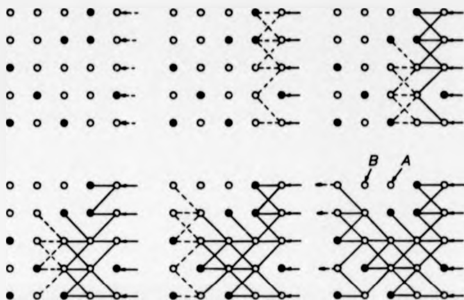


Figure 5.5: Step-by-step generation of AVAILability signals showing avoidance of dead-end routes.

only if the cell has at least one TRUE incoming AVAILability signal. This procedure has the effect of flushing out the dead end routes from the array. Any dead end route will result in all the cells on that route outputting FALSE AVAILability signals. This is illustrated in figure 5.5 which shows the column by column movement of AVAILability signals across the array. The faulty cells (marked in black) are obviously un-AVAILable; however, in addition, some of the functional elements are shown as being un-AVAILable. The cell labelled A is un-AVAILable since all of its right hand neighbours are faulty, while that labelled B is un-AVAILable because although not all right hand neighbours are faulty, none of them are AVAILable.

In the second stage of the process REQuest signals are generated starting from the left hand column and working from left to right across the array. This is illustrated in figure 5.6 in which we consider an input REQuest being applied only to the uppermost cell in the left hand column which is outputting a TRUE AVAILability signal. Further REQuest signals are then generated in subsequent columns by following a path of TRUE AVAILabiltiy

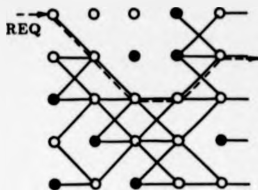


Figure 5.6: Generation of REQuest signals for first functional row.

signals according to the rules given in table 5.2. In this way a complete functional row can be constructed since a cell outputting a TRUE AVAILabilty signal indicates that a continuous path exists between the left and right hand sides of the array.

Subsequent functional rows are generated by alternate cycles of AVAILability and REQuest generation until all possible functional rows have been formed. This procedure ensures that any processors which have become unavailable as a result of the presence the first functional row will output the appropriate AVAILability signals.

5.3.6 5-Neighbour *WINNER* algorithm

In the foregoing description of the *WINNER* algorithm only nearest-neighbour interconnections between adjacent columns were permitted. This resulted in a cell which could communicate with three neighbouring cells in both the column to the left and the right. However, this restriction is not due to any fundamental limiting property of *WINNER* and the algorithm can be extended in a straightforward manner to a scheme which allows a cell to communicate, for example, with any of 5 neighbouring cells in the column to the left or right.

The main advantage of a 5-neighbour interconnection scheme is that configuration performance will be improved. This is due to the longer range

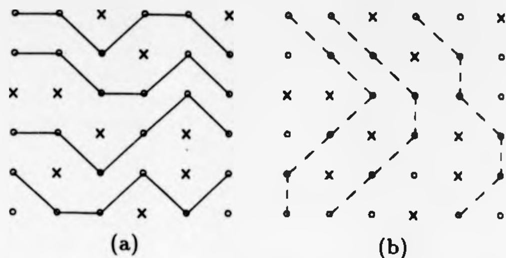
communication which has been introduced which allows some cells to be used in the configured array which would otherwise have been omitted from the array. The truth tables for the control logic can be extended in the obvious manner to handle the extra inputs and outputs and will not be described in detail.

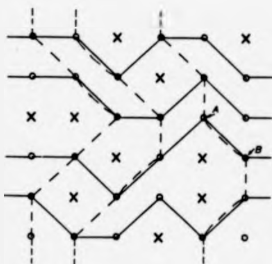
5.4 Algorithm 2: *WINNER* In Two Dimensions

Algorithm 1 makes the basic assumption that it is always possible to bypass faulty cells in the vertical direction. In many cases this may be a valid assumption and the technique could be used successfully in many applications. However, when we started this work our basic philosophy was that the configured array should completely avoid all faulty processors. In this section we will show how the one dimensional *WINNER* algorithm can be extended to encompass this philosophy by applying it to both the rows and the columns simultaneously, so that bypassing of the faulty processors is avoided. To see how this is done the reader is referred to figure 5.7.

Figure 5.7(a) illustrates a typical small array which has been configured in the horizontal direction using the *WINNER* algorithm in 1 dimension. Functional rows have been generated which each contain a number of working processors equal to the width of the original array. Figure 5.7(b) shows the same array which has been configured in the vertical direction using the same one dimensional *WINNER* algorithm but this time operating on the columns. Here, columns are constructed which keep as close as possible to the left hand side of the array and each functional column is equal in length to the height of the original array.

Since the configurations generated by the algorithm consist of full width rows and full height columns, one might suppose that the superposition of the rows and the columns would generate an array of functional processors at the points of intersection of each row with each column, and that the





(c)

Figure 5.7: The *WINNER* algorithm applied in two dimensions: (a) Configured rows, (b) Configured columns, (c) Superposition of the functional rows and columns to create a functional array.

processors within the array region would have orthogonal interconnections. The simple superposition of separately configured rows and columns is shown in figure 5.7(c), where the processors forming the final array are indicated in black. Some processors have either horizontal or vertical connections but not both. These are discussed in section 5.4.1 and are controlled to act as bypasses in the direction in which they have connections. The size of the functional array is determined by the number of functional rows and columns generated. If p is the number of functional rows and q the number of functional columns, then simple superposition should generate a functional array of dimensions $p \times q$. A cell suitable for use with this algorithm would have extra communication channels and control signal paths for both the horizontal and vertical directions and is illustrated schematically in figure 5.8.

At first sight this simple superposition process appears to be quite straight forward. However, there are in fact two types of undesirable condition which can occur when the rows and columns are superimposed in such a simple manner. These have been called *double site* and *crossover* conditions and must be handled within the algorithm if a correctly configured array is to be produced for all fault distributions. The example illustrated here contains several double sites. As we shall see, a simple technique has been developed which results in an algorithm capable of configuring a functional array from any fault distribution.

5.4.1 Double Site Condition

Referring to the small array configured by simple superposition of functional rows and columns shown in figure 5.9(a) we see that there are pairs of processors, for example A and B, which both belong to the same functional row and the same functional column. The effect of this is that there are two processor sites, A and B, where only one is required. The problem can be overcome by instructing one of the processors to act as a bypass in both the horizontal and vertical directions. In our implementation we always instruct

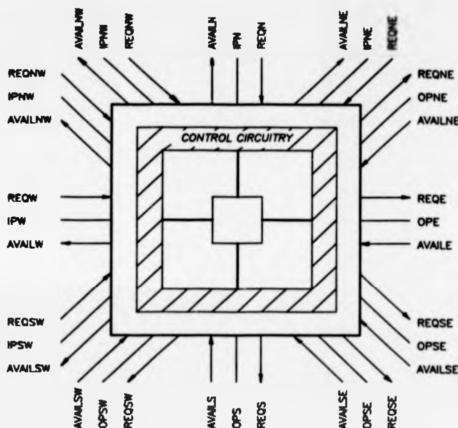


Figure 5.8: Schematic of cell suitable for 2-dimensional *WINNER* algorithm.

the upper processor to become the bypass although the lower processor could be chosen equally. The rule for doing this is as follows:

> If a cell finds that it is REQuesting to be connected to a cell to its SW or SE in both its row and its column configuration algorithms, it will become a bypass for its horizontal and vertical connections.

At first sight it may appear that since the process of avoiding double sites requires functional processors to be discarded the functional array size might be reduced as a result. However, these processors could not have formed part of the functional array anyway and discarding them does not affect the array size of $p \times q$.

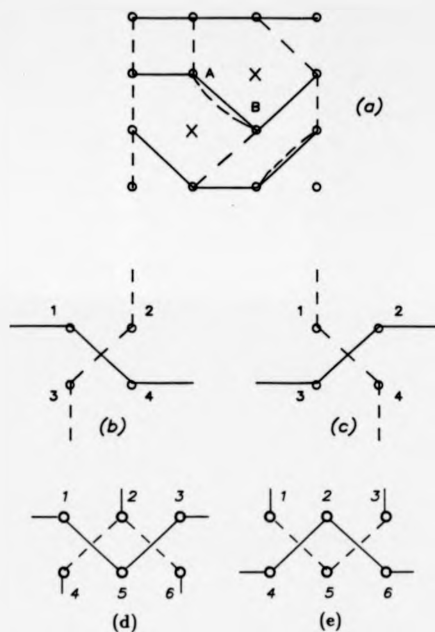


Figure 5.9: Double-site and crossover modes which can occur with simple superposition of functional rows and columns: (a) Double-site condition, (b) Fundamental crossover mode 1, (c) Fundamental crossover mode 2, (d) Composite crossover mode, (e) Composite crossover mode.

5.4.2 Crossovers

Crossovers occur when a row and a column intersect each other at a point other than a processor site. Crossovers can occur in two distinct ways as shown in figures 5.9(b) and (c). Unlike the double site condition which can be overcome without altering the configured rows and columns, the solution to the crossover condition requires either the functional row or functional column containing the crossover to be physically altered so that the superposition crossover does not occur. Alteration of the row or column may of course produce disturbances which propagate throughout the array until a new stable configuration is achieved.

The two fundamental modes in which crossovers can occur are shown in figure 5.9(b) and (c). Other crossovers can occur, such as those shown in figure 5.9(d) and (e) but these are simply superpositions of the fundamental modes. In mode 1, figure 5.9(b), the crossover could be avoided by making cell 3 unavailable to cell 2 in the presence of the link between cells 1 and 4 (link 1-4). Alternatively, cell 4 could be made unavailable to cell 1 in the presence of link 2-3. In a similar mannder, mode 2 crossovers, figure 5.9(c), can be avoided by making cells 3 or 4 unavailable in the presence of links 1-4 or 2-3 respectively.

We have proposed two solutions to the crossover problem, either of which can be embodied within the final two dimensional *WINNER* algorithm. The first requires an additional single-bit control signal to pass between cells in the East-West and North-South directions while the second requires a change only to the logic of the control circuitry in each cell.

Crossover Avoidance using Extra Control Communication

This technique requires the use of an extra, single-bit control signal which must pass between cells in both the horizontal and vertical directions. This allows the rows to be generated as in the one-dimensional *WINNER* algorithm but restricts the generation of columns to sites which will not cause

crossovers. As we have seen from figure 5.9(b), the crossover could be avoided if cell 3 was made unAVAILable to cell 2, and in figure 5.9(c), the crossover could be avoided if cell 4 was made unavailable to cell 1. The first of these can be achieved by using a single extra bit which propagates between cells from North to South. The bit indicates whether or not the cell which generated it is outputting a row REQuest in the SE direction and if so causes the column AVAILNE signal in the cell below to be inhibited. The second crossover mode may be avoided in a similar manner by passing an extra bit from West to East, indicating whether a cell has output a row REQuest to the NE, and if so inhibiting the column AVAILNW in the cell to its right. The cost of this technique is two extra inputs and outputs plus two (A AND NOT B) logic functions to perform the inhibitory action.

Crossover Avoidance by Alteration to Control Circuitry

The ideal solution to the crossover problem would be one involving a simple alteration to the control circuitry in each cell without requiring any extra connections between cells, since this is likely to introduce the smallest overhead of area into the algorithm. An approach incorporating these concepts has been developed and is now described.

To avoid mode 1 crossovers we wish to make cell 3 unavailable to cell 2 in the presence of link 1-4. This can be achieved without using extra control bits by noting that if link 1-4 exists, then because this is the highest priority direction for the row generation circuitry in cell 4, cell 4 will output a FALSE AVAILabilty signal to cell 3. If the 1-4 link does not exist, then cell 4 outputs a TRUE AVAILabilty signal to cell 3. This means that the 1-4 link can be detected by cell 3 by the value of the AVAILabilty signal coming from cell 4. The AVAILability signal can therefore be used to modify the AVAILability of cell 3 in the NE direction of the column generation circuitry, ie to cell 2. This technique can be implemented by ANDing the NE column AVAILability output signal of each cell with the incoming Eastern row-AVAILability signal. This gives the row link priority over the column link, which must then find

an alternative route.

In a similar manner, mode 2 crossovers can be avoided. In this case, a FALSE column AVAILability output from cell 4 to cell 2 indicates that the link 1-4 is present. This information can then be used in cell 2 to inhibit (with a single AND gate) its row AVAILabilty to cell 3. An alternative row route will then have to be found.

5.5 Advantages of the *WINNER* Approach

The *WINNER* self-organising approach to the configuration of 2-dimensional processor arrays is quite different from other published techniques and has several distinct advantages as follows:

- The array can configure itself automatically without the need for external assistance,
- The self-organising algorithm is fully convergent and cannot become unstable,
- As we shall see in chapter 7, the control circuitry associated with each cell in the array is simple, about 20 gates, resulting in a low hardware overhead,
- The technique results in good utilisation of the functional processors particularly when processor yield is greater than 80%,
- No global control lines are required,
- The array is potentially capable of being *re*-configured in the event of an in-service failure and could therefore be useful for remotely sited equipment, or in equipment requiring a very fast repair time.

5.6 Concluding Remarks

The self-organising algorithms presented in this chapter form the basis for this thesis. The algorithms have so far been described at a high level of

operational detail. In later chapters we develop the ideas more fully and cover logical and statistical simulations, testing, hardware requirements and describe a system which has been built to demonstrate the ideas.

Chapter 6

Performance of the WINNER Algorithm

6.1 Introduction

In chapter 5 we described a novel self-organising algorithm called *WINNER* for providing fault tolerance in 2-dimensional processor arrays. In this chapter we address the task of evaluating the performance of *WINNER* for different sizes of array, processor yield and overhead. We also consider a technique which could be used to increase the performance of the *WINNER* algorithm for large processor arrays. The technique involves partitioning an array into a number of groups of columns. We then consider the configuration approaches proposed by other authors as described in chapter 4 and compare their performances with that of *WINNER*.

6.2 Simulation

The ideal method by which to evaluate and compare performances of different configuring algorithms designed to tolerate faults in integrated circuits would be to fabricate chips which have been designed using the techniques and observe how well the techniques tolerated real defects. The final proving of a concept must be done in this way, but initial comparisons can be achieved in a much more efficient manner by computer simulation. Simulation is not a perfect tool for evaluating fault tolerant techniques since it is

extremely difficult to generate a precise model of the defects introduced by the fabrication process. Particularly difficult in this respect are faults affecting global circuitry such as power supplies and clock lines. Furthermore, the distribution of defective processing elements can only be estimated since as we saw in chapter 3, this varies from process to process. However, even in the presence of these difficulties, simulation offers high levels of user interaction and flexibility at relatively low cost and remains the most important tool used in the literature for evaluating algorithms. In this thesis, simulation is used as the main basis for evaluation and comparison of algorithm performance.

6.3 Choice of Language

The ultimate aim of this research into self-organising algorithms is to produce hardware suitable for use in the design of large integrated circuits and possibly even Wafer Scale Integration. It is therefore possible in principle to use a hardware description language (HDL) such as ELLA to evaluate the performance of *WINNER*. However, although ELLA is ideal for simulating a single array at the hardware level, and can provide essential information about the signal levels existing in the array, it is not particularly suited to simulating large numbers of arrays with different fault distributions to provide statistical information about the algorithm. However, by describing algorithms in a behavioral manner rather than in hardware form it is possible to use a serial programming language (SPL) for simulation purposes.

SPLs have a number of advantages over HDLs for statistical simulation of the type to be carried out, as follows:

1. The flexibility of SPLs means that changes in the algorithms, parameters or statistical requirements can be easily made by simply changing parameters of the program; in HDL's such changes can often produce many consequential changes.

2. The simulations require many different random fault distributions around

which to configure an array; these are readily generated in SPL's but not in HDL's.

3. With SPL's, the scope for analysing the results of a simulation within the program are almost unlimited, whereas in HDL's, there is almost no opportunity even for counting the number of rows which have been configured.

4. The efficiency, in terms of the CPU time required is usually better in SPL's than in HDL's partly because of the way in which the rules are specified; in SPL's the rules can be simpler because they are described at a higher level.

5. The amount of memory required for SPL's is much smaller than that for HDL's since the HDL representation of the algorithm has all the connections between elements of the array explicitly included for all elements for all time. In SPL's, the connections between array elements are represented within loops, and only one single connection exists at any one time.

For these reasons, the statistical simulations were carried out using the SPL, Algol68.

6.4 Simulation Requirements

The first step in the task of simulating the fault tolerant algorithms is to decide what aspects of the algorithms are to be evaluated. It is also important that the results can be sensibly compared with equivalant results produced by other algorithms. A characteristic which has become popular in the literature is the performance of the algorithm in utilising the working processors in the array. This is frequently termed the *Harvest*, and is defined as the fraction of the total number of functional cells which have been used in the configured array; it is usually expressed as a percentage. The concept

of a harvest enables an estimate to be produced of how well an algorithm has performed since it is directly related to the number of cells which are potentially available for use in configuring an array. For example, in an array containing 10 rows and 10 columns of processors with a 50% yield, only 50 processors are of any use, and it is the percentage of these which can be configured that provides the figure for the harvest.

It is the view of the author, however, that although the harvest does provide a handle into algorithm performance, it is not the most useful characteristic for evaluation purposes. In some algorithms, the curve of harvest against processor yield for a particular array size and overhead decreases monotonically with decreasing processor yield. For other configuration algorithms, in particular those based on nodal fault tolerance, this is not the case. For Hedlund's 4-out-of-12 nodal fault tolerance scheme the relationship between harvest and processor yield for a 10 by 10 array with 100% overhead has been evaluated by Franzon (1986) and is shown in figure 6.1. When the processor yield is 100%, the harvest is 33.3%, since only a third of the processors are theoretically being used. The introduction of a single fault anywhere in the array causes the harvest to rise since the same functional array can now be constructed from fewer functional processors. As the processor yield is reduced the harvest eventually starts to fall since rows containing many functional processors become unusable.

However, the main interest of a potential user of a configuring algorithm is not the harvest, since it does not tell him directly what size of array he will need in order to achieve the required array size with a particular probability. For this reason, the work in this chapter is based on configuring *target* arrays of various sizes, with the number of spare rows of cells required to enable the target array to be configured being evaluated for a range of processor yields. From this information, a potential user can immediately deduce the size of array he will need for his application.

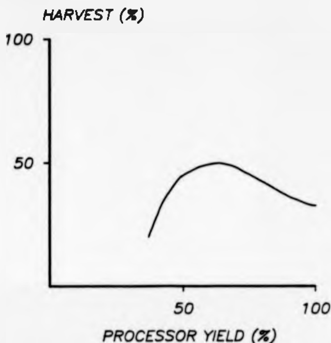


Figure 6.1: Harvest of the scheme of Hedlund (1982).

6.4.1 Program Parameters

A program to perform the required simulation has been written and generates a table of results as illustrated schematically in figure 6.2. The program can be run for each different target array size required. The table of results consists of a two dimensional array of numbers and is essentially a yield map for the appropriate target array as a function of processor yield and rows of overhead, in the x and y directions of the table respectively. Each result in the table is an average over many arrays having identical overhead and processor yield, but differing random fault distributions.

Several program parameters can be varied as follows:

1. Size of target array,
2. Range of number of rows of overhead,
3. Range of processing element yield,

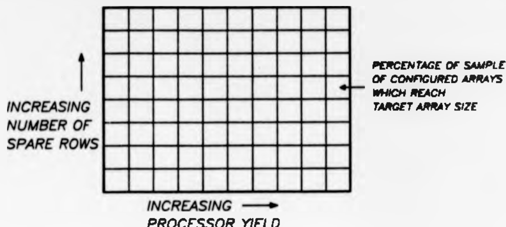


Figure 6.2: Schematic of the table of simulation results.

4. Number of samples (for statistically significant set).

These parameters are set at the start of each program run.

6.4.2 Program Flow Chart

Initially, a program to carry out the simulation was written in a fairly straightforward manner which involved calculating the result for each point in the result table. However, it became clear that large amounts of CPU time were being consumed and it was necessary to optimise the program to reduce run times so that large arrays could be simulated. It was noted that the tables of results had a characteristic form similar to that shown schematically in figure 6.3. As can be seen, all the useful information in the table is contained within a fairly narrow band bounded by a region of zero array yield on the left and 100% array yield on the right. It was therefore clear that most of the CPU time is spent calculating predictable values and that significant time savings could be made if only values within the band were evaluated. The problem is that the position of the band within the table is unknown at the start of simulation.

This problem has been overcome by using a program whose flow chart is shown in figure 6.4. The essential feature is that the program first searches

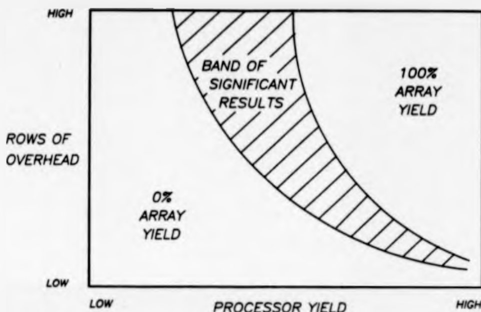


Figure 6.3: Characteristic form of the table of results.

for the band, and when found, evaluates all entries within the band using a recursive procedure. The procedure detects the edges of the band by noting the first occurrence of a 0% or 100% array yield, and uses this information to avoid doing further unnecessary calculations. The program listing has been included in Appendix A for the benefit of the interested reader.

For a typical table, the CPU time has been reduced to less than a quarter of that used when all table positions were evaluated.

6.4.3 Square Arrays

A 2-dimensional processor array can have arbitrary numbers of rows and columns. However, it is not possible to simulate all combinations of rows and columm and in the absence of a requirement for a specific size of target array, it was decided that it would be best to simulate a number of different sizes of square target array. However, the simulation program is quite general and could be used for arrays of any dimension if desired.

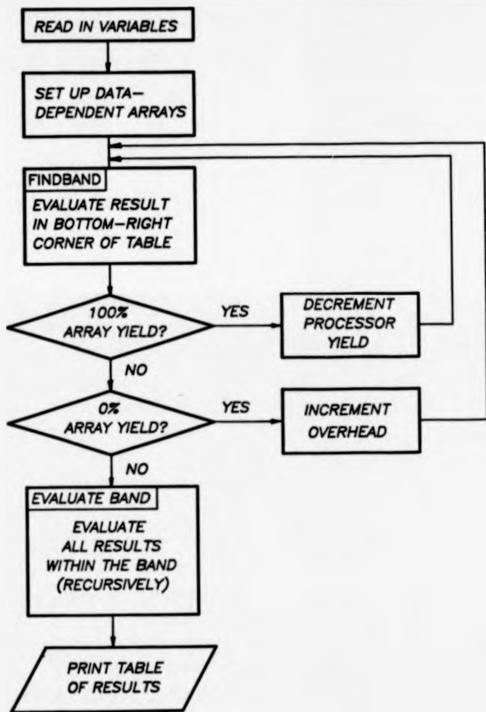


Figure 6.4: Flow-chart of the simulation program.

6.5 *WINNER* Simulation Results

Full tables of results have been generated for a range of target array sizes as follows:

- 4 by 4,
- 5 by 5,
- 6 by 6,
- 8 by 8,
- 10 by 10,
- 12 by 12,
- 16 by 16.

Above target array sizes of 16 by 16, the CPU time required to generate a full table of results becomes very large, (for example > 1CPU day). For this reason, partial tables have been generated for:

- 18 by 18,
- 20 by 20,
- 32 by 32

Each target array size has been evaluated with an overhead between 0% and 200%. This range was chosen since it was felt that an overhead of more than 200% (ie 3 times the circuitry of the target array) was probably generally unacceptable. The partial tables of results for target arrays larger than 16 by 16 contain results for 200% overhead only.

A typical table of results is shown in figure 6.5. This is actually for a 10 by 10 target array, but other sizes of array are similar in shape. A number of useful graphs can be drawn from the information in each table as well as from the relationships between tables. These graphs are discussed in the following section.

OVERHEAD (ROWS)	60	64	68	72	76	80	84	88	92	96	100
15	3.5	32.5	76.0	98.0	.	.	.	.	.	.	.
14	6.5	16.5	71.0	94.0	.	.	.	.	.	.	.
13	.	11.5	56.0	88.5	98.5	.	.	.	.	.	.
12	.	7.5	41.5	80.0	98.0	.	.	.	.	.	.
11	.	4.0	22.0	65.0	94.5	99.0	.	.	.	.	.
10	.	1.5	9.0	54.5	89.5	99.0	.	.	.	.	.
9	.	.	4.0	31.5	81.5	97.5	.	.	.	.	.
8	.	.	2.0	15.0	59.0	92.5	99.0	.	.	.	.
7	.	.	.	4.5	36.0	86.0	97.0	.	.	.	.
6	.	.	.	3.0	15.5	54.5	91.5	.	.	.	.
5	.	.	.	.	2.5	36.5	80.0	99.0	.	.	.
4	.	.	.	.	.	6.5	43.5	89.5	.	.	.
3	.	.	.	.	.	.	6.0	67.0	97.0	.	.
2	.	.	.	.	.	.	.	14.0	61.0	.	.
1	.	.	.	.	.	.	.	.	7.5	71.5	.
0	.	.	.	.	.	.	.	.	.	.	.

PROCESSOR YIELD (%)

Figure 6.5: Table of simulation results for a *WINNER* array.

6.5.1 Graphical Presentation of *WINNER* Results

Several families of curves can be drawn from the tables of results produced by the simulation program:

1. Array yield as a function of processor yield for various values of overhead,
2. Overhead as a function of processor yield for various values of array yield,
3. Array yield as a function of target array size.

These are described in detail in the following sections.

Variation of Array Yield with Processor Yield

For each size of target array a curve can be drawn of the array yield achieved as a function of processing element yield for different values of overhead. The families of curves for target arrays of 5 by 5, 10 by 10 and 16 by 16, configured using the *WINNER* algorithm with 3-neighbour connectivity are shown in figure 6.6. With a processor yield of 100% every array can be configured to produce the target array. However, as the processor yield is reduced, the array yield remains at 100% until a critical level of processor yield is reached. At this point the array yield begins to drop rapidly. For an overhead of 30%, ie 3 spare rows in the 10 by 10 target array, the critical processor yield is about 95%, and the array yield becomes virtually zero at a processor yield of 85%. Arrays with larger percentage overheads have lower values for critical processor yield. The main feature to note from these curves is the steepness of the fall from 100% array yield to 0%. This means that a very small change (say 1 or 2 percent) in processor yield can have a very significant effect (say 10 or 20 percent) on the array yield.

A similar family of curves can be drawn for each size of target array. Each family is similar in form, but the position of the curves in the x direction moves to the right for larger arrays, indicating poorer array yields as target array size increases. This feature is investigated in a later section.

Overhead as a Function of Processor Yield

Curves of the overhead required as a function of processor yield to achieve different values of array yield are probably the most useful to a potential user. Such curves are essentially contour maps of the tables of results generated by simulation. In practice they have been produced from the curves of array yield against processor yield described in the previous section since these curves allow interpolation between the relatively coarse points of the table.

Figure 6.7 shows the curves for a 10 by 10 target array with contours of constant array yield of 10%, 50% and 90%. Curves for other values of array

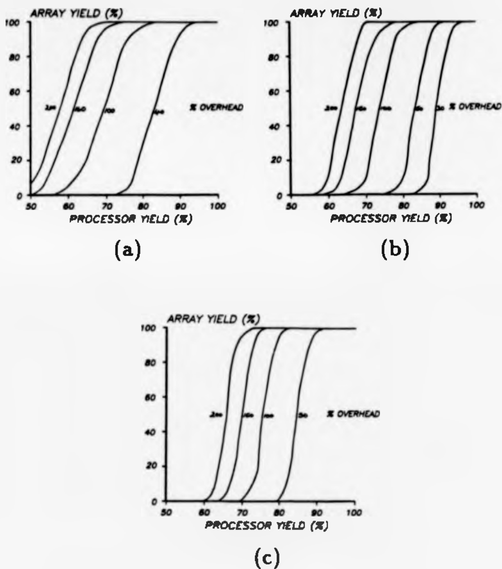
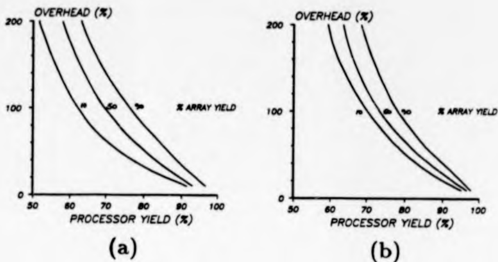


Figure 6.6: .Array yield as a function of processor yield for different values of processor overhead: (a) 5 by 5 array, (b) 10 by 10 array, (c) 16 by 16 array.



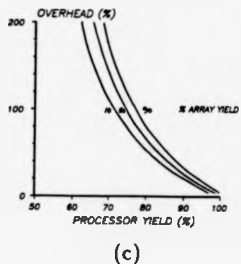


Figure 6.7: Overhead as a function of processor yield for different values of array yield: (a) 5 by 5 array, (b) 10 by 10 array, (c) 16 by 16 array.

yield can easily be drawn but have been omitted for clarity. The curves can be used to estimate the size of array which would be necessary to generate a 10 by 10 target array for a given processor yield. As an example, if a 50% average array yield is required and the processor yield is 80%, then it can be seen that an overhead of 50% is required, resulting in a starting array of 15 rows by 10 columns. Conversely, the required processor yield can be estimated from given values of array yield and overhead.

A family of similar curves can be produced for each size of target array and the relationship between these different families is the subject of the next section.

Variation of *WINNER* performance with array size

An important result which has emerged from simulating a number of different sizes of array is that for any given element yield and percentage overhead, the array yield becomes less for larger array sizes. This can be seen from figure 6.8 which shows the processing element yield required to achieve array yields of 10%, 50% and 90% as a function af array size. As can be seen, all the curves show that for a particular array yield, an increased processing element yield is required as array size is increased. However, the gradient of the curve does reduce rapidly with increasing target array size.

6.6 Improving *WINNER* Performance

In this section we consider the effect of two techniques designed to improve the performance of the basic *WINNER* algorithm. Both techniques involve increasing the connectivity of the cells in the array so that greater scope for avoiding faulty cells is available. The first involves increasing the number of neighbours with which each cell can communicate from 3 to 5 as described in chapter 5. In this technique, apart from each cell having more neighbours to choose from during configuration, the *WINNER* algorithm operates exactly as in the 3-neighbour case. The second technique involves partitioning an

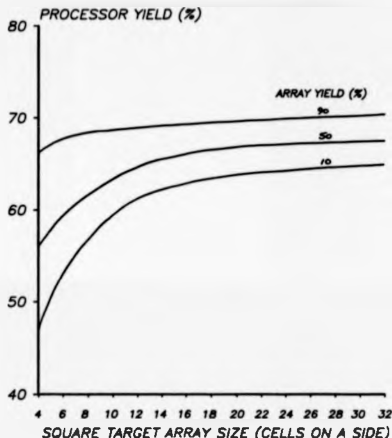


Figure 6.8: Variation of array yield with target array size.

array into several groups of columns which are configured separately and then joined by a longer range communication network.

6.6.1 5-Neighbour *WINNER* Algorithm

The 5-neighbour *WINNER* algorithm has been simulated in exactly the same manner as the 3-neighbour algorithm and the results are shown in figure 6.9. Figure 6.9(a) shows the relationship between array yield and processor yield for a 10 by 10 array with various levels of overhead, while figure 6.9(b) is a contour map of constant array yield as a function of processor yield and overhead. The corresponding performance of the 3-neighbour algorithm is shown dotted for comparison. As can be seen, at 100% overhead, and 50%

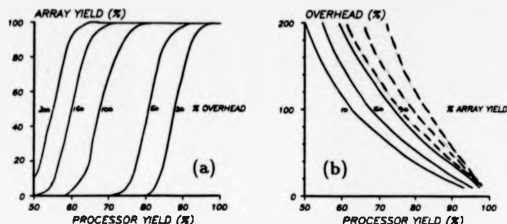


Figure 6.9: 5-Neighbour *WINNER* simulation results: (a) Array yield as a function of processor yield, (b) Overhead as a function of processor yield.

array yield, the 5-neighbour *WINNER* algorithm requires a 67% processor yield whereas the 3-neighbour algorithm requires a 73% processor yield. At 200% overhead, the required processor yields are 55% and 63% respectively.

6.6.2 Array Partitioning

Although the shape of the curves in figure 6.8 indicates that higher processing element yields or greater overheads will be required to achieve a given array yield as the size of the target array is increased, the results can also be interpreted in a more optimistic way. Suppose we want to generate an N by N target array with a certain yield. We can estimate the overhead and element yield which would be required to achieve this. According to figure 6.8, however, these figures can be reduced if we generate four target arrays with dimensions $N/2$ by $N/2$, and butt them together to produce the required array. The curves tell us that using this partitioning approach, the N by N target array can be generated with lower processing element yield. Indeed, we could generate nine $N/3$ by $N/3$ arrays to provide even greater advantage.

In practice there is little advantage to be gained by splitting the initial array in the horizontal direction, since the same result is achieved by par-

titioning the array into groups of columns. Each of the groups of columns is then configured as usual and the blocks connected together using routing circuits between each block. The routing circuits simply join the N functional rows of one block to the N functional rows of the neighbouring block and can be designed so that the self-organising ability of the entire array is maintained across the partitions. The actual circuitry required for this is presented in chapter 7. With more and more partitions, we eventually end up with single columns and this is the *ideal* row generating algorithm, but requires a large amount of circuitry to interconnect the configured groups of colmuns. There is therefore a trade off between array yield and number of partitions and this is investigated in the following section.

The reason that the partitioning procedure provides an advantage over the straight self-organising algorithm is that it introduces a degree of longer range communication into an otherwise nearest-neighbour communication algorithm. This allows certain previously intolerable fault distributions, to be tolerated by allowing faults to be avoided using long range communication.

The relationship between array yield and array width has been simulated for a target array of 12 rows with groups of columns of width 2, 3, 4, 5, and 6. The results have been plotted in figure 6.10. It can be seen that for say 50% array yield we need about 65% element yield for the 12 by 12 array but only 48% element yield for an array containing 2 columns block. At first sight this sounds like a tremendous improvement since in theory, by placing six 12 by 2 arrays side by side and routing between them we could produce a 12 by 12 array from a much lower element yield. In practice, of course, the routing circuitry does not have a 100% yield itself, and the increase in array yield will be modified by the routing circuit yield. The following section considers the overall advantage which could be gained.

Effect of Partitioning

In this section the effect of partitioning on array yield is examined with the yield of the column interconnection circuitry being taken into account. The

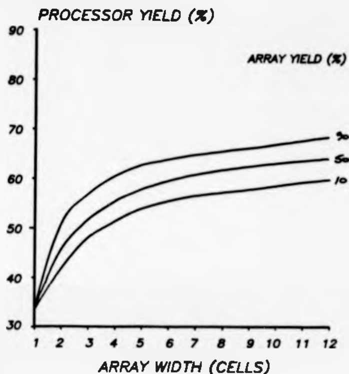


Figure 6.10: Array yield as a function of array width for a target array of 12 rows.

results are presented in graphical form in figures 6.11 and 6.12. Figure 6.11 shows how the results are obtained, and is described below, while figure 6.12 compares the results for various values of starting yield. All the results are for 200% overhead and illustrate the relationship between overall array yield and the number of groups of columns into which the array is partitioned.

In figure 6.11, curve A shows the increase in array yield which would be achieved if the extra circuitry (column interconnection circuitry and and control circuitry) had a perfect yield (ie 100%). The curve has been drawn by taking an arbitrary value of element yield, in this case 65%, and noting from the table of simulation results the value of array yield achieved (in this case 50%). The other points on the curve have been found by looking at the simulation results of arrays which have been partitioned into 12x6, 12x4, 12x3, and 12x2 arrays, and noting the new array yield for the same processor

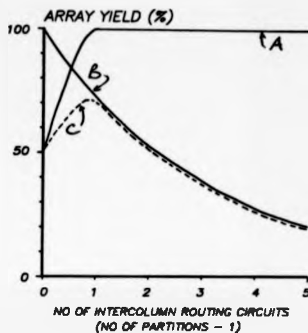


Figure 6.11: Effect of partitioning on array yield for a 12 row target array.

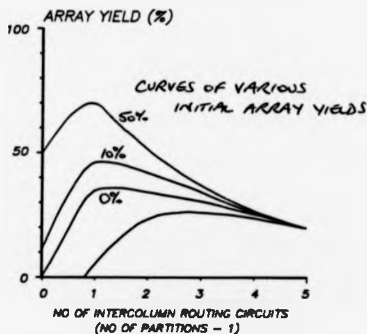


Figure 6.12: Effect of the number of partitions on array yield.

yield of, in this case, 65%. In other words we are observing the change in array yield for a fixed element yield as we reduce the partition size.

As shown by curve A, the use of two partition blocks, causes the array yield to increase from 50% to almost 100%. Obviously the introduction of further blocks cannot increase the array yield any further and the curve is flat for these values. The effects of the yield of the column interconnection circuitry results in further curves as follows. Curve B shows the estimated yield of the column interconnection circuitry which acts as a reducing factor on the array yield. The number of column interconnection circuits increases with the number of partitions, and its yield therefore drops exponentially as shown[1]. The yield of the control circuitry in each cell is high and since, as we shall see in chapter 9, it is possible to employ techniques to mask the effect of control circuit faults from the rest of the array, little degradation of the array yield will result from such faults. The net array yield is the product of the yield of the interconnection circuitry and the basic array yield and is shown in curve C.

As can be seen, the introduction of one partition, ie using two block each being half the width of the original target array, provides an improvement in array yield from about 40% to about 55%. However, increasing the number of partitions does not improve the array yield further because the progressively poorer yield of the routing circuitry begins to dominate, and the initial increase in array yield is gradually eroded. The results of figure 6.11 illustrate how one curve C has been generated, but of course a whole family of curves of this type can be drawn for different element yields. Several of these curves are presented in figure 6.12.

The most likely application of this approach is in arrays which have a very small or even zero array yield. In these cases, production may be non-viable unless the array yield can be increased. As can be seen, the worse the initial array yield, the more partitions are required before the peak in the array yield is reached. If the element yield is well below that required to achieve

[1] This assumes that the column interconnection circuitry is not fault tolerant.

a non-zero array yield, several partitions are required before any increase in array yield is achieved, but the increase actually achieved is proportionally greater.

6.7 Comparison with other Algorithms

In this section we compare the performance of the *WINNER* algorithms (3-neighbour and 5-neighbour) with the performance of other published configuring algorithms. Comparisons are made with the following algorithms:

1. Simple row-bypassing,
2. Triple Modular Redundancy (TMR),
3. The 4-out-of-12 nodal fault tolerance scheme of Hedlund and Snyder (1982),
4. The scheme of Moore and Mahat (1985),
5. The best row generation scheme theoretically possible; Sami and Stefanelli (1983)
6. The best global configuration scheme theoretically possible.

Contours of constant array yield for each of these algorithms will be plotted as a function of overhead and processor yield. The overhead required to achieve a particular array yield for a given processor yield can then be taken as a measure of algorithm performance.

6.7.1 Bounds on Performance

When evaluating the performance of any system it is useful to have estimates of highest and lowest possible performances even if these can only be achieved in theory. These limits are called the upper and lower bounds. Bounds on the performance of configuring algorithms can be determined as follows.

Lower Bound on Performance

The lower bound on performance for configuring a two-dimensional array clearly occurs when the array is unable to tolerate any faults and as a result will have zero array yield for any processor yield of less than 100%. This is a rather trivial bound.

Upper Bound on Performance

The upper bound on configuration performance is of more interest. In fact two bounds are relevant to us as follows:

1. Upper bound on performance of all possible configuration schemes,
2. Upper bound on performance of row generation schemes.

The first of these can be calculated in a simple manner since it occurs when all functional processors in the array are used in configuring the target array. The fractional overhead is given by

$$Overhead = \frac{1 - Y_P}{Y_P} \tag{6.1}$$

where Y_P is the processing element yield. This formula represents an absolute upper bound and cannot be exceeded.

The upper bound on the performance of row generation schemes occurs when any row containing at least as many functional processors as there are columns in the required target array is considered to be a functional row. The functional rows are then interconnected in an appropriate manner to form the functional array.

Assuming that an N row by N column target array is required, that the fractional overhead is k and that the probability that a processor is functional is p, we can use the binomial distribution to find the probability that a row is functional, $P(Row)$, as follows

$$P(Row) = \sum_{i=N}^{kN} {}^{kN}C_i (1 - p)^{kN-i} p^i \tag{6.2}$$

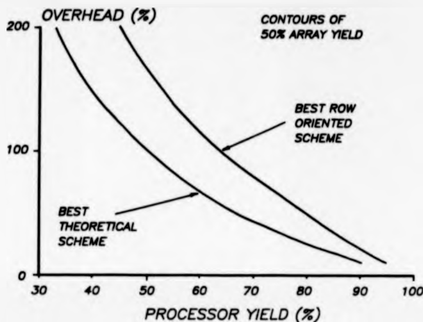


Figure 6.13: Upper bounds on array yield for a 10 by 10 target array as a function of processor yield.

giving an array yield of

$$P(Array) = [P(Row)]^N \tag{6.3}$$

where

$$^{kN}C_i = \frac{(kN)!}{i!(kN-i)!} \tag{6.4}$$

The upper bounds calculated above are shown in figure 6.13.

6.7.2 Algorithm Performance Comparisons

Each of the configuring algorithms has been simulated under the same conditions as the *WINNER* algorithm so that a fair comparison can be made. The results are presented in graphical form in figure 6.14 which shows the overhead required to achieve a 50% array yield for a 10 by 10 target array, as a function of processor yield. The upper performance bounds have been included for reference.

As can be seen from figure 6.14, the row-bypass scheme, TMR and Hedlund's scheme perform rather poorly. The performance of the row-bypass

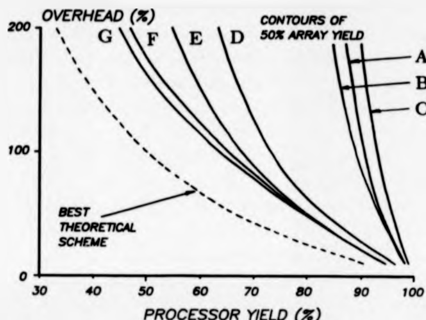


Figure 6.14: Comparison of the performance of various configuring schemes. A: TMR, B: Hedlund's scheme, C: Row bypass scheme, D: *WINNER* 3-neighbour, E: *WINNER* 5-neighbour, F: Moore scheme C, G: Sami and Stefanelli's scheme.

scheme is poor because it is a very simple algorithm and requires very few switches and interconnections in its implementation. TMR is also a very simple scheme but suffers because the minimum possible overhead is 200%. Hedlund's scheme requires configuration of the block of 12 processors and also has a minimum overhead of 200%. It appears to offer little advantage over TMR.

Of the remaining configuring schemes simulated, it can be seen that scheme C of Moore and Mahat (1985) performs better than the *WINNER* algorithms, although the difference between the Moore scheme and the *WINNER* algorithm with 5 neighbours is not significant at processor yields above about 70%. The reason for this is that both schemes have the same degree of connectivity, ie 5, but whereas the communication path lengths in *WINNER* are limited to two cells, the Moore scheme permits any communication length to be used. This enables the Moore scheme to perform better than

WINNER at lower processor yields. However, long communcation paths can cause serious delays in high performance processor arrays, and Moore and Mahat suggest in their paper that the communication lengths could be restricted if desired. This would reduce the performance of their scheme and bring it closer to that of the 5-neighbour *WINNER* algorithm for processor yields below 70%.

The best row generation algorithm is that of Sami and Stefanelli (1985) and essentially represents the upper bound of row generation schemes. The algorithm involves spare columns rather than spare rows and considers any row containing sufficient functional cells to form a row of the target array to be a functional row. Such functional rows are then interconnected using a bus oriented scheme. Although the results for this algorithm are presented for an overhead range of 0 to 200%, the cost in terms of switches and interconnections quickly becomes impractical and in practice overheads of a few columns would be the maximum contemplated.

At processor yields above about 80%, the difference in performance between the *WINNER* algorithm with 3 neighbours, 5 neighbours or the Moore scheme is small, and in the absence of any other constraints, the scheme with the simplest hardware implementation should be chosen.

6.8 Conclusions

In this chapter we have evaluated the performance of the *WINNER* algorithms and compared them with competing techniques which have been published in the literature. From the simulations it has become clear that the techniques of row-bypassing, Triple Modular Redundancy and Hedlund's 4-out-of-12 nodal fault tolerance scheme all have a very poor performance compared with the other schemes simulated. The scheme of Sami and Stefanelli (1983) is clearly the best but least practical scheme. For processor yields above about 80% there is little to choose between the Moore and Mahat (1985) scheme C, and the two different *WINNER* schemes. Below 80%

processor yields, Moore and Mahat's scheme or the 5-neighbour *WINNER* scheme should be chosen.

Chapter 7

Hardware Implementation of the WINNER algorithm

7.1 Introduction

In chapter 5 we described several related algorithms which could be used to enable a 2-dimensional processor array to organise itself around faulty processors and generate a functional array. The purpose of this chapter is to consider the hardware implications of the approach. This will include the hardware requirements for the control circuitry, and the circuitry required for entering and removing data from the configured array. Where appropriate we develop formulae relating complexity to the number of data signal lines passing between cells. We also discuss the simulations of hardware which have been carried out to verify correct operation of the circuits. We restrict our study to the 3-neighbour *WINNER* algorithm applied to one dimension, the rows, of the array. This limitation has been imposed because as we have seen from chapter 6, the one-dimensional algorithm offers the best performance and is therefore the most likely algorithm to be used in practice. The extension of the hardware to the 5-neighbour algorithm is simple.

We then consider the hardware required to implement other configuring schemes proposed in the literature, and compare these with *WINNER*.

7.2 Hardware Requirements of *WINNER* Control Circuitry

In this section we consider the hardware requirements for implementing the control circuitry for the *WINNER* algorithm applied in one dimension. We first consider the gate level implementation to obtain a circuit which is independent of technology and then consider how this circuit could be translated into a CMOS transistor-level design and develop a formula for the number of transistors required to implement it.

7.2.1 Gate-Level *WINNER* Control Circuitry

The gate level implementation of the *WINNER* control circuitry is shown in figure 7.1. We have assumed for simplicity that only a single data line passes through the cell in the vertical and horizontal directions. The circuit can be readily extended to multiple data lines as will be described later. The circuitry has been divided into two distinct parts, as follows.

1. Decision making logic,
2. Data routing logic.

The decision making logic communicates with neighbouring cells via the REQuest and AVAILability signals and eventually decides which data connections should exist between which cells. The data routing circuitry is shown shaded, while the remainder of the circuitry forms the decision making logic.

The decision making logic is essentially a direct implementation of the truth tables for REQuest and AVAILability generation given in chapter 5 with the assumption that the pass/fail indication would be provided by a mechanism such as self-test as described in chapter 8. The data routing circuitry selects the data associated with an incoming REQuest signal and applies it to the input of the processor. Since the algorithm prevents more than one REQuest being sent to any cell, this function can be implemented

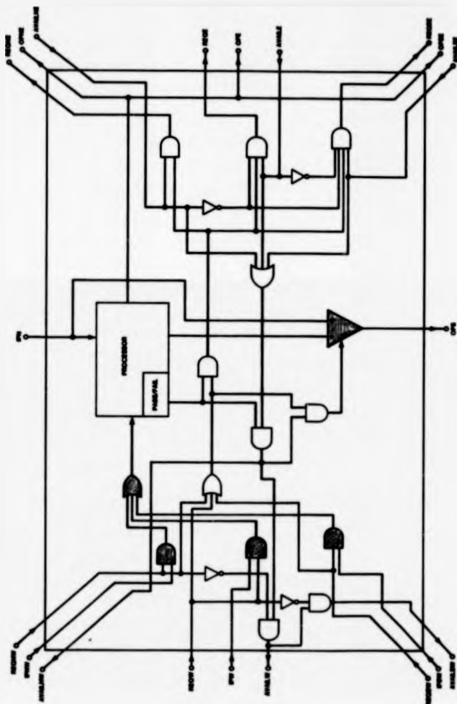


Figure 7.1: Gate implementation of the control circuitry required to perform the 1-dimensional *WINNER* algorithm.

in the simple manner shown, where incoming REQuests are ANDed with their respective data lines and the outputs of the AND gates are ORed together to produce the required processor input. In the vertical direction, the processor is bypassed by the multiplexor circuit unless the cell contains a functional processor AND has at least one TRUE REQuest AND one TRUE AVAILability input. This means that cells containing faulty processors and cells which are unused are omitted from the configured array.

The complexity of the above circuitry is 18 gates counting each element in the circuit as a single gate. This figure, however, has limited significance because the basic building block in integrated circuits is the transistor and the number of transistors per gate varies depending on the technology in which it is to be implemented. For this reason the next section considers the transistor level circuit that would be required in a CMOS implementation.

7.2.2 Transistor Complexity of Control Circuitry

The CMOS implementations of the gates used in the circuit of figure 7.1 are shown in figure 7.2. A direct translation of the *WINNER* control circuitry into transistors can be made by replacing each gate with its transistor-level equivalent. A formula for the number of transistors can now be developed. The complexity of the circuit can be expressed as:

$$Number\ of\ transistors = T_D + T_x L_x + T_y L_y \tag{7.1}$$

where T_D is the number of transistors required for the decision circuitry which generates REQuest and AVAILability signals, T_x and T_y are the number of transistors required for routing respectively each horizontal and vertical data line and L_x and L_y are the number of horizontal and vertical data lines respectively which must be routed through the cell, and depends on function of the processor.

From figure 7.1 and 7.2 the values of T_D, T_x and T_y can be determined as $T_D = 60$, $T_x = 6$ and $T_y = 6$, giving:

$$Number\ of\ transistors = 60 + 6L_x + 6L_y \tag{7.2}$$

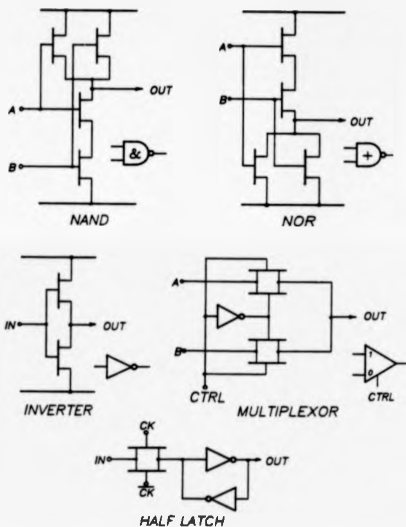


Figure 7.2: CMOS implementation of the gates used in the *WINNER* control circuitry.

In a real design it is likely that the circuit would be optimised for the particular technology to be used in the fabrication process. This may involve using NAND and NOR gates wherever possible instead of AND and OR gates. However, it is not appropriate to carry out an optimisation of this type since as far as possible we require results which are independent of technology.

7.3 Input/Output Interface Circuitry

A characteristic of the one-dimensional *WINNER* algorithm is that functional rows are generated extending from one side of the array to the other. This means that access to the ends of the rows is easy since they reside at the edges of the array. However, the precise positions of the end of the functional rows depends on the distribution of faulty processors in the array and will not generally be known in advance. From the user's point of view this is unacceptable since he wishes to apply his inputs and receive the outputs from fixed sets of I/O pins. As a result, interface circuitry for both inputs and outputs has been designed to automatically perform the mapping of a fixed set of data inputs onto a spatially variable set of functional rows, and vice-versa for the array outputs.

7.3.1 Selecting Functional Rows

We assume that the data flow through the array is from left to right and from top to bottom. This does not limit the applicability of what is to follow, but makes its description more straightforward.

From the description of the *WINNER* algorithm presented in chapter 5 we remember that REQuest signals pass from left to right across the array and AVAILability signals pass from right to left. On the left hand side of the array, the end of each functional row will be marked by a TRUE AVAILability signal emerging from the western output of a cell. Similarly, on the right hand side of the array, a TRUE REQuest signal from the eastern output of a cell

marks the other end of a functional row. The ends of the rows on the left and right hand sides of the array will have the same spatial ordering, but in most cases will have different spatial positions. The presence of TRUE REQuest and AVAILability outputs in the positions described could therefore be used by an interface circuit to control the routing of data into and out of the configured array.

In addition to performing the data routing task described above, the input interface circuitry should be able to detect whether sufficient rows have been configured and if not, inform the user. It should also be possible for the input interface circuitry to disable surplus functional rows if more have been configured than are required.

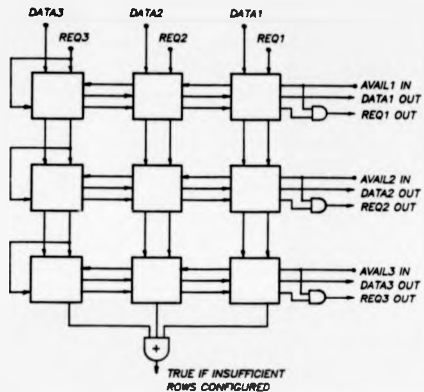
We now describe circuitry suitable for providing the input and output interface functions.

7.3.2 Data Input Circuitry

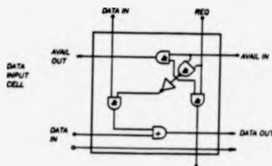
A data input circuit suitable for ensuring that data signals are input to the appropriate rows of the *WINNER* array is illustrated in figure 7.3(a). It consists of an array of simple, identical cells whose function is shown in figure 7.3(b) together with a single AND gate per row to provide the REQuest inputs to the array. The height of the data input array is equal to the number of rows in the *WINNER* array while the width is equal to the number of functional rows required in the configured array. The REQ inputs to the top of the data input array are all set to logic 1.

Operation of the Data Input Circuitry

The data input circuitry operates in the following manner. The value of REQ indicates the status of the data input with which it is associated, being TRUE in cells through which the data passes before being routed to a functional row and FALSE thereafter. Similarly, the AVAILability signal entering the data input array from the *WINNER* array indicates the status of the row of the configured array with which it is associated. It is TRUE in a cell of a



(a)



(b)

Figure 7.3: Data input circuitry. (a) Data input array, (b) circuit of single cell.

row of the data input array only if the corresponding row of the *WINNER* array is functional and an input data line has yet to be routed to it. It is FALSE in all other cells in the row. Once set to FALSE, the REQ and AVAILability signals cause cells of the data input array to take no further part in the routing of data into the configured array.

The right hand column of the data input array will route its data down the column until a cell with a TRUE AVAILability input is reached. This will be at a position corresponding to the start of the first functional row of the configured array. Within this cell the input data is routed to the functional row via the pass transistor, and both the REQ and AVAILability output signals from the routing cell are set to FALSE. This means that no other data input will be connected to the same functional row and that the current data input will not be connected to any other functional row. A REQuest input to the *WINNER* array is generated by the AND gates on the right hand side of the array whenever a functional row is detected. The routing of the other data inputs takes place in a similar manner.

Insufficient Functional Rows

It is possible that the distribution of faults in the array is such that fewer functional rows are configured than are required. In this case, one or more of the data inputs will not be routed to a functional row. The shortfall will manifest itself in the data input circuitry as one or more REQ signals which remain TRUE when the emerge from the bottom of the data input array and may be detected by the OR gate shown on figure 7.3(a). If the output of the OR gate is TRUE, at least one of the required functional rows could not be configured.

Surplus Functional Rows

In the same way that some fault distributions may result in insufficient functional rows being generated, others may result in a surplus. These surplus rows must be avoided so that they do not interfere with the function of the

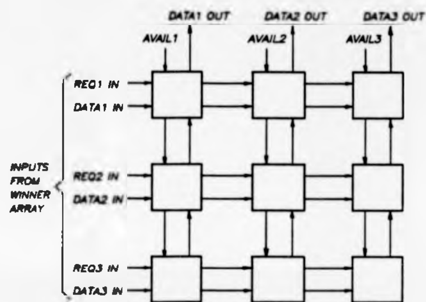
required rows. In chapter 5 we saw that when the *WINNER* algorithm is applied to one dimension of the array, cells would be controlled to act as bypasses in the vertical direction if the processor within the cell is faulty or if no REQuest inputs are received by the cell, ie if the cell is unused. Surplus rows can therefore be avoided by sending TRUE REQuest signals only to the required number of functional rows. This is achieved automatically by the data input array by feeding the REQ input arriving vertically at each cell in the left hand column of the data input array into the horizontal REQuest input in the same cell, as shown in figure 7.3(a). The REQ signals passing between cells in the data input array are then TRUE until all data inputs have been routed to a functional row. Thereafter, the REQ signal is FALSE and inhibits REQuests being applied to the *WINNER* array.

7.3.3 Data Output Circuitry

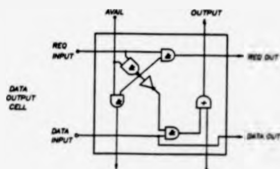
The data output circuitry required to map the spatially variant data outputs from the right hand end of functional rows onto a fixed set of output lines is shown in figure 7.4(a). As can be seen, the circuit is in the form of an array similar to that used for the data input array. The cell function is shown in figure 7.4(b) and it will be noticed that it is in fact identical to the cell used in the input array (if the straight through REQuest line is removed from the input cells), and is simply rotated anti-clockwise through 90 degrees in the array. The operation of the data output array is identical to that of the input array. Data emerging from the first functional row is therefore routed to the first column of the output array, and so on.

7.4 Column Interconnnection Circuitry in Partitioned Arrays

As was described in chapter 6 the overall array yield can be increased by careful partitioning of the processor array into groups of columns. The columns are then configured separately and joined together to form the final array.



(a)



(b)

Figure 7.4: Data output circuit. (a) Data output array, (b) circuit of a single cell.

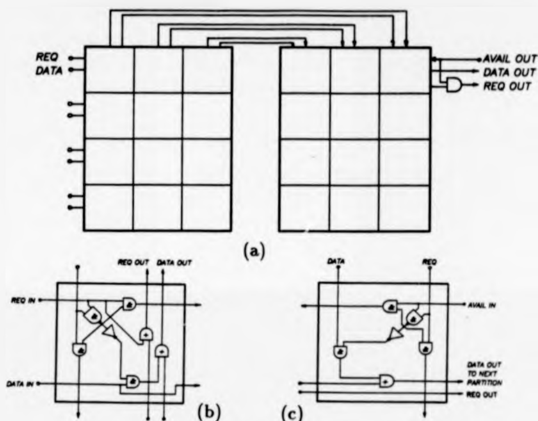


Figure 7.5: Intercolumn routing circuitry. (a) Intercolumn array, (b) single cell of left hand part of array, (c) single cell of right hand part of array.

The circuitry required to perform the interconnection of the columns is shown in figure 7.5(a) with the cell circuitry being shown in figure 7.5(b). The operation of the column interconnection circuit is very similar to that of the data input and data output circuits described in the previous sections. Essentially, the left hand array in figure 7.5(a) is identical to the data output array, except that a REQuest output accompanies each output signal. The data and REQuest outputs from the functional rows of one group of columns emerges from the top of the array and are fed into the second array as shown. The second array is simply a data input array which routes the signals to the appropriate functional rows. True AVAILability signals are fed to all rows of the left hand group of columns.

Since the column interconnection circuitry transfers the REQuest signals from one group of columns to the other, all the features of the self-organising algorithm are maintained across the partition, including the handling of surplus and insufficient functional rows.

7.5 Simulation of the *WINNER* Hardware

In this section we describe the simulation of the hardware used in the *WINNER* cell. The hardware description language, ELLA was used and this is briefly described first.

7.5.1 The ELLA Hardware Description Language

ELLA is an acronym for Electronic Logic LAnguage and was developed at RSRE Malvern (Morison et al, 1982), and is currently being marketed by Praxis Systems Limited of Bath, UK. It is a hardware description language (HDL) together with a simulator and design environment. Like other HDLs ELLA allows the user to describe circuits in a programming language and then animate them by running the simulator. During simulation, inputs can be applied to the circuit being simulated and the responses generated by the circuit can be observed. This means that circuits can be checked for satisfactory operation and mistakes corrected before any effort is put into physical design of the circuit. In order to enable circuits with unexpected behaviours to be analysed, the inputs and outputs of every node in the circuit can also be observed as required. This is the equivalent of being able to observe any point of a breadboard circuit using a logic probe. ELLA is a particularly suitable language for use in describing the circuits used in *WINNER* because it has very powerful instructions which allow regular arrays to be described simply and in an elegant fashion.

7.5.2 Simulations using ELLA

All of the circuits proposed for the *WINNER* algorithm have been described and simulated in ELLA and have operated as expected. An array of both the one and two dimensional *WINNER* cells has been simulated and a number of different fault distributions applied. A correctly configured array was generated for all distributions of faults.

The benefits of including all the programs written for the purposes of simulating circuits described in this thesis are limited. However, the ELLA program written to describe the one dimensional *WINNER* algorithm is presented in Appendix B for the interested reader.

7.6 Hardware for other Configuring Techniques

As we found in chapter 6, the main competitors to the *WINNER* algorithms were the configuration techniques proposed by Moore and Mahat (1985) and Sami and Stefanelli (1983). In this section we consider the hardware which would be required to implement these techniques and compare the results with that needed in *WINNER*.

7.6.1 Moore and Mahat's Scheme

Moore and Mahat proposed three configuration schemes, A, B and C. Schemes B and C have the best performance and will be used here for comparison purposes. From figure 4.9 it can be seen that scheme B requires 5 switching elements per cell while scheme C requires 8 per cell for each data line passing between cells in the horizontal direction. In addition, each switch will require a latch to store the switch position. We also assume that full latches will be used and interconnected serially so that switch control data can be clocked serially into a shift register.

Assuming that a full latch requires 12 transistors, and that each switch requires 2 transistors, the circuitry required in each cell to perform the

routing function is as follows.

$$Number\ of\ Transistors(Scheme\ B) = 60 + 10L_s \quad (7.3)$$

$$Number\ of\ Transistors(Scheme\ C) = 96 + 16L_s \quad (7.4)$$

7.6.2 Sami and Stefanelli's Scheme

Although the configuration scheme proposed by Sami and Stefanelli (1983) is not likely to be practical for more than a few spare rows, a general formula for the complexity of the routing circuitry has been derived. The connectivity requirement in the algorithm is for a fully connected network whose height is equal to the height of the array, and whose width is equal to the number of spare rows in the array. This results a routing complexity as follows.

$$Number\ of\ Transistors = 12N_S + 2N_S L_s \quad (7.5)$$

As can be seen, both terms depend on the number of spare rows, N_S. The equation can be rewritten as

$$Number\ of\ Transistors = N_S(12 + 2L_s) \quad (7.6)$$

indicating that the circuitry enclosed in brackets is required per cell for each spare row used.

7.6.3 Comparison of Hardware Requirements

The formulae derived in previous sections have been presented graphically in figure 7.6, which shows the complexity per cell needed to implement each configuration scheme as a function of the number of horizontal data signal lines passing between cells. In the case of Sami and Stefanelli's scheme, several curves for different numbers of spare rows are shown.

It can be seen that the scheme of Sami and Stefanelli requires the least hardware when 1 or 2 spare rows are used. However, with 4 spare rows, the

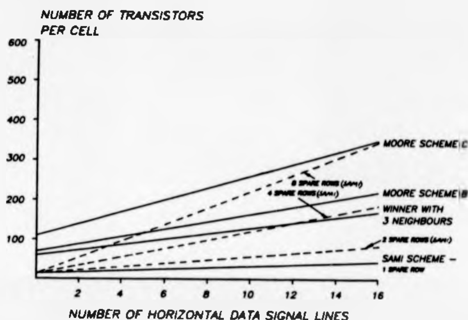


Figure 7.6: Comparison of the hardware requirements of various configuration schemes.

basic 3-neighbour *WINNER* scheme becomes more attractive, requiring less hardware than any of the other schemes.

The fact that *WINNER* requires less hardware than most other schemes is an interesting result, since the other schemes also require external control of the switches, whereas *WINNER* achieves fully automatic decision making and data routing. This result was quite unexpected and arises mainly because no switch control information needs to be stored in the cell since it is generated by the cell itself.

Chapter 8

Self-Testing of Self-Organising Arrays

8.1 Introduction

In Chapter 5 when describing the operation of the self-organising techniques based on the *WINNER* algorithm it was assumed that some method existed by which the processor in each cell of the array could reliably indicate whether or not it was functional.

Techniques for self-test are well known in the literature and have been used in the design of production devices, for example the Motorola microprocessor range (Daniels and Bruce, 1985). Self-testing approaches have evolved from earlier work on improving the manual testability of circuits. In this chapter we describe the motivation behind the goals of design for testability and of self-test and present an established approach to both problems. In chapter 9 these approaches (with slight modification) will be applied to the testing problem in *WINNER*.

8.2 Design for Testability

It is now widely recognised that whilst VLSI technology offers many advantages in terms of processing power per Watt or per square centimetre, it also generates numerous problems concerning the testability of the circuit so created. This stems partly from the fact that the circuits contain many

more components which all have to be checked and partly because the ratio of input/output pins on chips usually decreases as the complexity of the chip increases (Landman and Russo, 1971). This means that access to the internal nodes of the chip can often become severely limited, resulting in at best long test times or at worst, incomplete testing of the device.

Many researchers are active in the field of *design for testability* which aims to improve external access to internal nodes of the chip by incorporating extra hardware for use during testing. This has an obvious cost in terms of hardware but the advantages it offers often outweigh the extra cost. Most approaches to achieving a testable design rely on a serial scanning arrangement to improve access to the internal components of the circuit. Two of these, the scan path and Level Sensitive Scan Design (LSSD), are now described.

8.2.1 Scan path techniques

One of the first examples of a scheme to increase testability was published by Kobayashi et al (1968). This paper is written in Japanese but describes what is known today as the *scan path*. The idea of the scan path is to allow data to be introduced and extracted from a circuit through a single pair of data lines so that the I/O overhead is kept to a minimum while maintaining a high level of controllability and observability. A scan path comprises a number of cascaded shift register elements each of which can receive data from either the output of the previous shift register or from a parallel input. The outputs of these elements are applied as test stimuli to the circuit under test, while, in parallel mode, the parallel inputs to the scan path are provided by the outputs of the circuit under test. A single scan path element and a block diagram of a scan path in position within a circuit are illustrated in figure 8.1. It is normal, but not essential for the circuit being tested by the scan path to be purely combinational. In sequential circuits it has been suggested by Williams and Angell (1973) that switches could be incorporated to change the circuit from *normal* mode to *test* mode. In test mode, the latches would

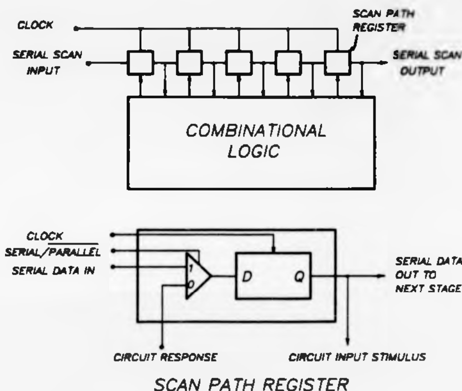


Figure 8.1: The scan-path testing technique.

be connected in the form of a serial shift register which would then be used in the same way as a scan path to test the remaining combinational circuitry. In both combinational and sequential circuits the test time can usually be reduced by using several independent scan paths which can then be used in parallel.

The scan path technique will be used in later sections as the basis for a control circuit testing strategy for use with the *WINNER* algorithm.

8.2.2 Level Sensitive Scan Design - LSSD

Another example of a technique which can improve the testability of a device is the Level Sensitive Scan Design technique or LSSD for short (Eichelberger and Williams, 1977). This technique formalises the scan path approaches for combinational and sequential circuits and has become a commonly used tool

in the design of circuits and systems.

8.3 Self-test techniques

The previous section has briefly outlined techniques for increasing the testability of circuits using serial scanning latches. These techniques form the basis for self testing circuits. The motivations behind self testing circuits are many. As circuit complexities increase, the number of test patterns required to fully test a circuit also increases, usually at a faster rate than the increase in number of gates in the circuit. This means that test times using serial scanning techniques can become unacceptably long. Furthermore, the increase in performance of circuits means that test equipment must be of the highest quality if tests are to be carried out at the rated speed of the device. Such test equipment is extremely expensive, and may soon become impossible to build to the required specification. On-board self-test can help in both of these areas.

The main features of a self test approach are illustrated in figure 8.2 and comprise the circuit under test together with a method of generating test stimuli and a method of compressing the results produced by the circuit when stimulated. It is of course possible to store a number of selected test patterns in a ROM and apply these to the circuit under test in a sequential manner. A ROM could also be used to store the expected responses from the circuit and compare them with the actual responses. Any differences could then be noted and used to pass or fail the circuit. This approach, however, would be very costly to implement since large numbers of test patterns are normally required for a full test. For this reason most self test techniques use an exhaustive sequence of test patterns which can be produced cheaply by a counter or Linear Feedback Shift Register (LFSR). In addition, test results are not individually compared with expected results but are *compressed* into a much reduced form which can be checked in a simple manner using a comparator. LFSRs and compressors are discussed in the following sections.

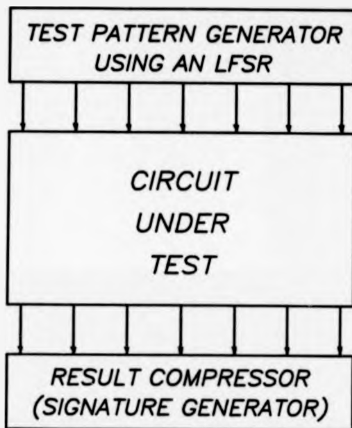
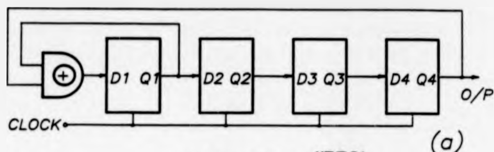


Figure 8.2: Block diagram of a typical self-testing system.

8.3.1 Linear Feedback Shift Registers

A typical LFSR is illustrated in figure 8.3(a). It comprises a number of cascaded shift register elements together with one or more exclusive-OR gates which feed information back from the outputs of some of the stages to the main input. When the shift register is clocked from any initial state (except all zeroes), a sequence of ones and zeroes can be observed at any point in the register, say its input, with delayed versions of the sequence appearing at sucessive register outputs. This sequence depends upon the initial state, the length of the register and the positions and number of feedback taps. The sequence produced by the register depicted in figure 8.3(a) is given in figure 8.3(b). It can be seen that all 15 possible states are achieved in the



	Q1	Q2	Q3	Q4	NEXT D1 INPUT
INITIAL STATE	1	0	0	0	1
	1	1	0	0	1
	1	1	1	0	1
	1	1	1	1	0
	0	1	1	1	1
	1	0	1	1	0
	0	1	0	1	1
	1	0	1	0	1
	1	1	0	1	0
	0	1	1	0	0
	0	0	1	1	1
	1	0	0	1	0
	0	1	0	0	0
	0	0	1	0	0
REPEATS FROM	0	0	0	1	1
HERE →	1	0	0	0	1

(b)

Figure 8.3: Generation of test patterns: (a) A 4-stage linear feedback shift-register (LFSR), (b) The patterns produced by the circuit in (a).

register at some point in the cycle, which then repeats. Shorter cycles can be achieved with different feedback taps, but for self-testing purposes we are generally interested in the longest, or maximal length sequences. The sequences produced appear to be random but are repeatable from a given initial state, hence the alternative name of the LFSR is the Pseudo-random pattern generator.

The LFSR is very suitable for use as a source of test patterns in self-testing circuits because it is a very simple structure, even simpler than a counter, which could perform a similar function. Because its output sequence is predictable it can be used in a deterministic way to generate expected results from the circuit under test.

8.3.2 Compression of test results

The motivation behind attempting to compress test responses emerging from the circuit under test is to reduce the cost (mainly in time) of scanning out large numbers of responses serially from the circuit for immediate checking by an external tester. The idea is to perform most of the evaluation of the responses within the circuit being tested. A compressed result is essentially a cumulative, short pattern, dependent on a long sequence of test responses.

Compression of test responses can be achieved by two main methods, namely counting and recursive compaction.

8.3.3 Compression by counting

The idea here is to count the number of some characteristic occurring in the sequence. Typical characteristics are the number of transitions, ie 0-to-1 or 1-to-0, or the number of edges, ie transitions in one direction. The assumption is that in a circuit containing a fault, the number of counted transitions will be different from that produced by a perfect circuit. The final count therefore provides a much reduced value which can be checked in a simple manner. Counting can be achieved using conventional counter circuits. The technique turns out to be inferior to the recursive compaction technique described in the following section and will not be further discussed.

8.3.4 Compression by Recursive Compaction

A circuit capable of performing recursive compaction is illustrated in figure 8.4. The reader will immediately notice the similarity to the LFSR pattern generator described in a previous section. The difference is that in order to provide an input for the serial data stream which is to be compressed, an additional exclusive-OR gate is included in the feedback path as shown. This extra input allows the incoming serial data stream to modulate the feedback to the first stage of the LFSR, which is then remembered by the shift register. For a given input sequence and given initial state, the same

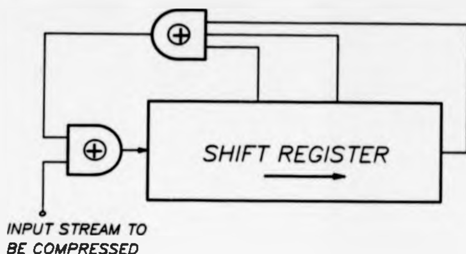


Figure 8.4: A recursive compaction circuit for single-bit input streams.

final pattern or *signature* will always be produced and can be used to detect streams which contain faults.

The use of the LFSR to compress data steams has been described by Frohwerk (1977), who also coined the term *Signature Analysis* to describe the approach when applied to circuit testing. Frohwerk shows that a single error in the incoming sequence will always be detected, and that the probability of non-detection when the number of faults is unrestricted is $1/2^n$, where n is the number of stages in the compactor LFSR. He also shows that counting techniques will be unable to detect faults with a probability of greater than $1/2^n$; clearly, recursive compaction is the method to be chosen.

8.3.5 Compaction of Multiple Input Streams

If the incoming data stream which is to be compacted consists of several serial streams, a slightly modified circuit is required. This is illustrated in figure 8.5, and shows exclusive-OR gates inserted between stages of the shift register in addition to the extra gate in the feedback path to the first stage. Each of these extra gates can accommodate a serial input data stream. The number of such streams is limited by the number of stages in the LFSR.

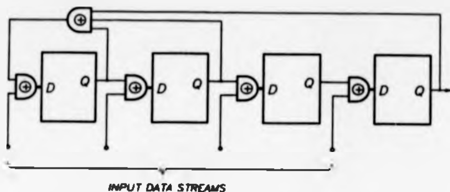


Figure 8.5: Recursive compaction of multiple input streams.

8.4 Self testing requirements in *WINNER*

Most of the literature on self testing describes techniques up to the point where the compressed value, or signature, of the response data streams is produced. This is because the motivation for self-test is primarily to reduce test time and increase the speed at which the test can be carried out so that it is more representative of the speed at which the circuit under test will operate when in service. For this purpose it is sufficient to read the signature by clocking it out of the circuit and comparing it, in an external tester with the expected value.

In *WINNER*, the self testing approach is to be applied separately to each processor in the array and the correctness or otherwise of each signature generated should ideally be determined by the corresponding cells themselves. We therefore need a circuit which can perform this function.

8.4.1 Signature Comparison

The comparison of the test signature with the expected value will involve the use of a comparator which has been preset (probably hardwired) with the expected signature. The output of the comparator (a single bit) will then provide a go/no-go indication which can be used by the control circuitry during the configuration of the array. A simple method by which this com-

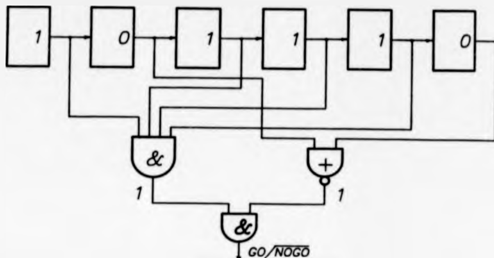


Figure 8.6: Simple signature comparison method.

parison could be achieved is illustrated in figure 8.6, in which the register outputs in which a 1 is expected are ANDed together, while those positions in which zeroes are expected are NORed together. The AND and NOR outputs are then combined in a second AND gate which provides the required results with a 1 indicating pass and zero indicating fail.

Chapter 9

Reducing Control Circuit Vulnerability

9.1 Introduction

In the foregoing descriptions of the operation of the *WINNER* algorithm we have described techniques for testing the processors within each cell of the array and how cells containing a faulty processor can be avoided by self-organising techniques. Throughout these discussions it has been assumed that the control circuitry operates correctly at all times. In general, however, this assumption will not be valid and in applications such as large area integration, it will almost certainly be necessary to take steps to identify faulty control circuits.

The purpose of this chapter is to address the problem of faults occurring in the control circuitry. The aim is not merely to detect the presence of faults, since this would result in the entire array having to be discarded, but to develop techniques by which the faults can be tolerated. As we shall see, this simply results in losing the opportunity to use the processor in the cell containing the faulty control circuit.

We present two quite different techniques for tolerating control circuit faults although both techniques exploit the same property of the *WINNER* algorithm as will be described. In the first technique, duplicated control circuits are used in each cell to enable faults to be detected and automat-

ically provides tolerance to many faults. The second technique is a novel approach involving the use of an external tester to mask out the effect of control circuit faults (Evans 1986, and Evans and McWhirter 1987). This approach can exhaustively test each control circuit and the inter-cell wiring and can guarantee that a correctly configured array has been produced. The hardware overheads associated with the techniques are also considered.

9.2 The Ideal Self-Organising Array

The ideal self-organising array would be one in which no external assistance is required during either the testing or configuration phases. The system should be able to detect any single or multiple faults present in the system and offer 100% confidence that if an array of the required size can be configured, that it is actually functional. If insufficient working processors are available, the system should be able to indicate this to the user.

In practice, the above requirements cannot be fully achieved because it is not possible for a system, none of whose component parts has been proven correct, to make guaranteed decisions. This is particularly true in the field of integrated circuits where faults will undoubtedly be present in the array. In such circuits we cannot rely on any part of the circuitry on the chip to perform its predefined task correctly. This means that in order to achieve 100% confidence in the circuit the manufacturer must perform at least a small test using an external, known-to-be good tester on some part of the circuitry. The tested circuitry, if found to be functional, can subsequently be relied upon and used in further tests of the wafer. The challenge is to develop a testing strategy which requires only a small amount of circuitry to be externally tested, and to be able to carry out the externally applied test in a simple manner.

9.3 Inherent Fault-Tolerance of the Control Circuit

The control circuitry used in the *WINNER* algorithm can be considered to have two types of output signal, either *active* or *passive*. This property arises from the fact that only TRUE outputs affect neighbouring cells in an active way, possibly resulting in the neighbour taking some positive action. FALSE outputs do not have any positive effect on neighbouring cells and are therefore considered to be passive outputs.

The property of having active and passive output signals means that the control circuits each have an inherent ability to mask some faults as now described. Since FALSE outputs have only a passive effect on neighbouring cells, any fault resulting in one or more stuck-at-0 faults on the control circuit outputs will not affect the rest of the array in a detrimental manner. The stuck-at-0 output error does not propagate beyond the boundaries of the cell containing the fault.

The proportion of single stuck-at faults which can be masked in this way has been shown by simulation to be 50%. Techniques for masking the remaining 50% of single and multiple stuck-at faults are described in the following sections.

9.4 Dual-Rail Implementation of Control Circuitry

In the basic *WINNER* algorithm a fault in the control circuitry could cause an array to be incorrectly configured if it produces an erroneous TRUE REQuest or TRUE AVAILability output. This would result in the entire array being discarded because there is no mechanism within *WINNER* which can tolerate such faults.

The probability of a fault in the control circuitry resulting in an entire array having to be discarded can be significantly reduced if duplicated control

circuits are used. This technique, sometimes called 'two-rail' implementation, (Wakerly, 1978), requires the use of two independent control circuits in each cell of the array. At its simplest, the idea is that if both circuits receive the same inputs they should generate identical outputs. If one of the circuits contains a fault, then for at least one input pattern, at least one of its outputs will be different from the corresponding output of the other, fault free circuit. This difference can then be detected.

In the context of the *WINNER* algorithm we need only to detect the erroneous signal and stop it propagating throughout the array. It is not necessary to correct the erroneous signal since, as we shall see, the self-organising capability of *WINNER* allows the cell containing the faulty control circuitry to be avoided.

There are two main variants of the two-rail implementation technique as follows:

1. Simple duplication of the control circuits,
2. The use of true and complement control circuits.

We now describe how these techniques can be used to detect errors in the outputs of circuits, and then show how the idea can be used in *WINNER* to enable faults in the control circuitry to be both detected and tolerated.

9.4.1 Fault Detection by Simple Duplication

In this approach two identical circuits are used in place of the single original circuit as shown for a simple circuit in figure 9.1. Each circuit receives identical input signals and in the absence of faults the two circuits should produce identical output signals. The possible outputs are as follows:

$$\left.\begin{matrix} 0 & 0 \\ 1 & 1 \end{matrix}\right\} \textit{no error} \qquad \left.\begin{matrix} 0 & 1 \\ 1 & 0 \end{matrix}\right\} \textit{error present} \tag{9.1}$$

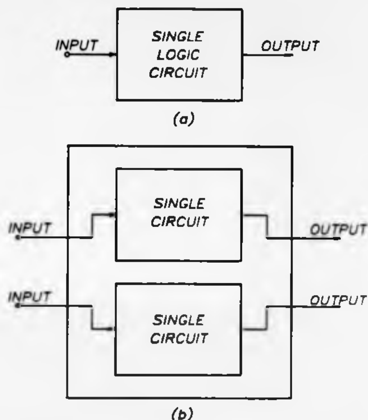


Figure 9.1: Fault detection by simple duplication: (a) Conventional circuit, (b) Two-rail implementation.

Any difference in the outputs from the two circuits can be detected using an exclusive-OR gate. If each circuit has more than one output line, corresponding pairs of output lines are compared using separate exclusive-OR gates. Once the circuits have reached a stable state an error in an output pair will be indicated by a TRUE output from the exclusive-OR gate. The outputs of the exclusive-OR gates could be ORed together to produce a single error indication if desired.

This approach will detect single or multiple faults provided that no fault present in one of the circuits produces the same error as any of the faults

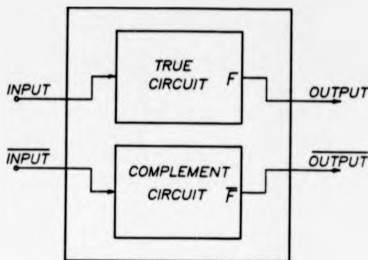


Figure 9.2: True and complement two-rail implementation.

present in the other. However, if the fault is due to a pair of output wires being shorted together, or failure of the power supply to both circuits, no fault will be detected since both output wires will carry identical signal levels.

9.4.2 Fault Detection using True and Complement Circuits

This approach avoids the problem of non-detection of shorted output pairs by requiring that one of the duplicated circuits is implemented in the normal manner to produce the required output function, F, but that the other is designed to produce the logically inverted output function, NOT F from logically inverted inputs as shown in figure 9.2. This means that a fault free pair of circuits will always produce output signal pairs each containing both a TRUE and a FALSE value. The possible output signals in each output pair are as follows:

$$\left.\begin{matrix} 0 & 0 \\ 1 & 1 \end{matrix}\right\} \textit{error present} \qquad \left.\begin{matrix} 0 & 1 \\ 1 & 0 \end{matrix}\right\} \textit{no error} \tag{9.2}$$

This type of two-rail circuit is preferred since it can detect unidirectional multiple errors such as those caused by loss of power, etc. In addition, the use of true and complement circuits means that there are no constraints on the layout of the circuit when being fabricated as integrated circuits, since faults common to both circuits will be detected.

The presence of an error can be detected as before by exclusive-OR gates which now will produce a FALSE output if the signals in a pair are identical. In both of these duplication techniques, the exclusive-OR gates can be placed either at the input to the circuit or at its output. In the former case, faults in both the previous circuit and the interconnecting wire can be detected, while the latter detects faults in the circuit but not those in the wiring.

Neither of the above approaches can *correct* errors since the error detection circuitry has no way of knowing which of the duplicated circuits produced the correct output. As a result of this faults can only be detected and not tolerated by these schemes.

However, when used in the context of the *WINNER* algorithm, the error detection capability of the duplicated circuits combined with the self-organising nature of *WINNER* enables many faults in the control circuitry to be both detected and masked automatically. The way in which this operates is described in the following section.

9.5 Application of Duplicated Circuits to *WINNER*

In the *WINNER* algorithm the outputs of the control circuitry are either active or passive as discussed in section 7.3. This property can be exploited in the two-rail implementation of control circuits to stop propagation of erroneous signals beyond the cell containing the fault. This can be achieved by using extra circuitry to convert two-rail input signals containing errors into passive input signals. In a normal control circuit, the active signal

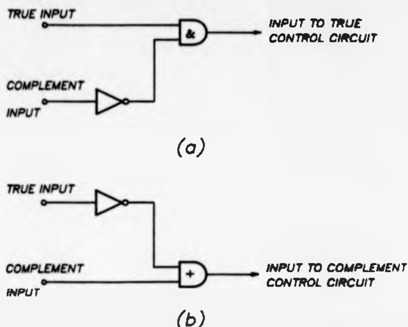


Figure 9.3: Two-rail input circuits for: (a) True *WINNER* control circuitry, (b) Complement *WINNER* circuitry.

level is TRUE, and the passive level is FALSE, while in a control circuit implemented as the complement of a normal circuit, the active and passive levels are reversed.

Assuming that true and complement circuits are used in preference to simple duplication, the inputs to the true and complement circuits for each value of the input pair are given in table 9.1. The two-rail signals can be converted according to table 9.1 for feeding to the true and complement circuits using the circuits shown in figure 9.3.

Input Pair Value		Error Status	Input to	Value of Input	
True	Comp		Control Circuitry	True	Comp
0	0	Error	Passive	0	1
1	1	Error	Passive	0	1
0	1	No error	Passive	0	1
1	0	No error	Active	1	0

Table 9.1: Conversion of Two-rail Control Circuit Input Signals.

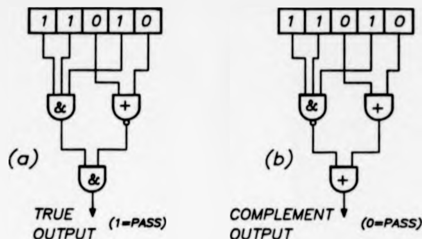


Figure 9.4: A two-rail signature comparator.

The signature analysis comparator can also be implemented in two-rail form and circuits to generate true and complement comparison outputs are shown in figure 9.4. These signals can be treated in the same way as the other two-rail inputs discussed above, since an error can be treated as a failing signature and assigned a passive value.

9.5.1 Performance of Duplicated Control Circuitry

It is difficult to accurately predict the effect that duplicating the control circuitry will have in a real circuit. However, the technique has been estimated by simulation with varying numbers of stuck-at faults as follows.

Both the single control circuit and the true and complement implementations have been simulated by applying a number of randomly distributed faults to the gates of the circuit with each fault being randomly stuck at 0 or 1. For each value of the number of faults, the circuit was simulated with 200 different distributions of faults and the number of distributions which caused one or more active errors in the outputs was counted and expressed as a percentage of the total sample. The results are presented in figure 9.5. It can be seen that for the single circuit implementation, 50% of single faults cause active output errors which would results in the entire self-orgainising

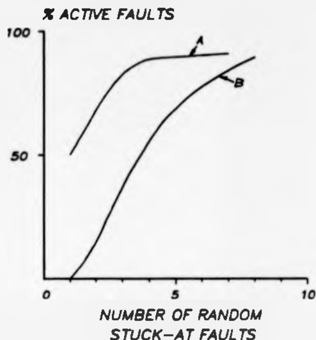


Figure 9.5: Percentage of active faults as a function of the number of faults occurring in the control circuit. Curve A: conventional implementation, curve B: dual-rail implementation.

array being discarded. By contrast, the true and complement implementation produces no active errors for any *single* fault. In both curves, the percentage of active errors rises rapidly with increasing numbers of faults, but the true and complement implementation can contain, on average, 3 random, stuck-at faults and still perform better than the single circuit.

Both curves, but particularly that for the single control circuit, saturate for large numbers of faults. This effect is believed to be due to the increased probability of a fault, which on its own would cause an active error, being masked by another fault which causes a passive error. For example, if a particular fault, deep in the circuit, causes an output to become stuck-at-1, ie an active error, a second fault could reverse the effect if it causes a stuck-at-0 error on the same output.

9.6 Hardware Requirements for Two-rail Implementation

In the two-rail implementation, two sets of decision logic are required. In addition, error detection circuitry on each input for both circuits will be required, amounting to 84 transistors. However, the data routing circuitry is unchanged, resulting in a cell complexity as follows.

$$Number\ of\ Transistors(Two-rail) = 204 + 6L_x + 6L_y \qquad (9.3)$$

This is about a factor of three greater than the single control circuit implementation. However, due to the significantly improved tolerance to faults, and the fact that the dual rail circuitry is still small compared to the processing element which are likely to be used, the technique could be very useful if problems of control circuit yield are encountered.

9.7 External Testing of the Control Circuitry

In this section we present a strategy based on (Evans and McWhirter, 1987) in which a scan path approach is used to detect faults occurring in the circuitry which has not been tested during the self-test procedure. We then show, with a small modification to the design of the scan path cell, how the effect of any faults detected can be masked so that the array can continue operating correctly. This approach provides a second level of fault tolerance in the *WINNER* technique.

9.7.1 Control Circuit Testing Strategy

The control circuitry associated with each element of the array is a purely combinational circuit and as such can be tested easily if it can be accessed from an external source. The required access can be provided by including a serial scan path between each column of processors as illustrated in figure 9.6 in which each dot represents a group of scan path registers in the AVAILability, REQuest and signal paths. This figure 9.6 also shows the position of

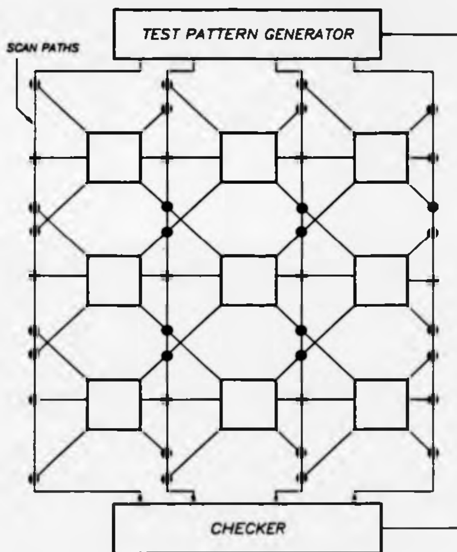


Figure 9.6: Schematic of the scan-path testing approach applied to *WINNER*.

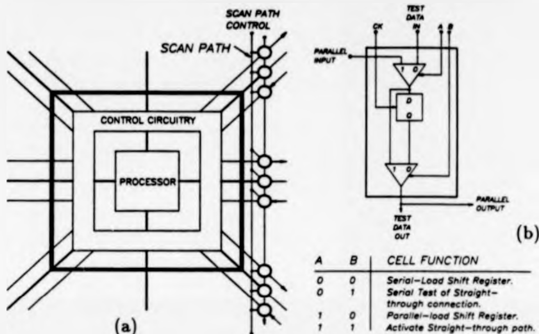


Figure 9.7: Scan-path arrangement within a *WINNER* cell: (a) Cell complete with scan-path, (b) Circuit of a single scan-path register.

the test pattern generator and checker which would both be external to the array and would be known-good-units. A cell complete with its scan path is illustrated in figure 9.7(a) with a single conventional scan path register being shown in figure 9.7(b). When the cells are joined together to produce the array a separate scan path is required on the left hand side of the array to provide the I/O access to the left hand column of cells as shown in figure 9.6.

As can be seen, the function of the scan register is controlled by two inputs, A and B. These signals determine the mode of the register according to the table shown in figure 9.7(b). Data can be clocked into the register from the output of the previous register stage, $(A = B = 0)$, or in parallel from the output of the circuitry under test, $(A = 1, B = 0)$. In addition, two other modes are available. One of these is used when the circuit is operating normally, $(A = 1, B = 1)$ and permits data signals to pass through the register cell from the output of one circuit under test to another without passing though a latched delay. This is achieved by using the straight-through

path, which bypasses the latch. The final mode, ($A = 0, B = 1$) allows the straight-through path to be tested from the external source. This is essential so that when the circuit is switched into normal operating mode after testing is completed, correct operation of the straight through path is ensured.

9.7.2 Scan Path testing procedure

The scan path testing approach operates as follows. Patterns are generated by the test pattern generator and with the scan register controls set to serial-load mode, are clocked serially into the scan paths. The outputs of the registers are connected directly to the inputs of the control circuitry and so the values in the registers are automatically applied as test pattern stimuli to the control circuits. After each pattern has been loaded, the scan path registers are set to parallel-load mode and the outputs of the control circuits are clocked into the registers. These results are then clocked out of the scan path serially and are checked for correctness. Any errors are noted and from the position of the error in the serial output pattern can be associated with the output from a particular cell.

The regular nature of the array and the simplicity of the control circuitry, mean that the number of test patterns required to perform an exhaustive test is small. The cell shown in figure 9.7(a) has seven input signals and therefore requires only 128 different test patterns. Since all the cells in the array are identical, the same test pattern can be used for every cell, and this further reduces the testing complexity.

The scan path procedure can detect faults within the control circuitry of the array and locate the fault to a particular cell in the array. This is the first requirement of a technique which can mask the faults from the rest of the array. The second requirement is for a technique to perform the masking process. This can be achieved using the approach now described.

9.7.3 Control Circuitry Fault Masking Procedure

As previously described, the design of the control circuitry is such that the outputs of the circuit are active only when at a high (logic 1) level. When any output is low, (logic 0), it has no effect on neighbouring control circuits and is termed passive. This property can be exploited to mask out cells containing faulty control circuits from the rest of the array if every output of the faulty control circuit is forced to zero, ie any active signal values are inhibited. The inhibit function must of course be performed by circuitry which is known to be fault-free so as to ensure correct execution of the masking process, and is described in the following section.

9.7.4 Modified Scan path Register

The inhibit function can be implemented using a scan path testing approach as described previously, but in which each scan path register has a small modification. A modified scan path register cell is illustrated in figure 9.8. When compared with the register previously used, it will be seen that the only difference is that the straight through path has been replaced by an AND gate, whose second input is supplied by the latch output. The effect of this is that when the straight through mode is selected, the value at the output of the register is determined not only by the parallel input signal, but also by the value in the latch. If the latch contains a high level, the circuit operates exactly as before, with the parallel input value passing directly to the parallel output. Alternatively, if the latch contains a low level, the AND gate will always output a low value, and therefore inhibit any active signal on the parallel input.

From the above description, it will be apparent that in order to mask out cells containing faulty control circuits, it is necessary to preload the latches, with zeroes being placed in registers corresponding to the outputs of the faulty circuit, and ones everywhere else. This can be achieved in the external test equipment by constructing a map of faulty cells, and generating

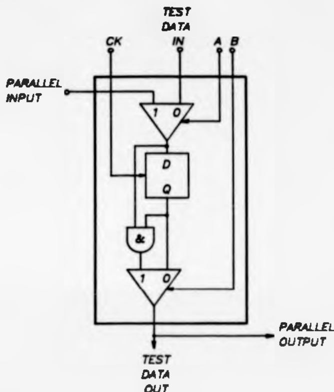


Figure 9.8: Modified scan-path register for use with *WINNER*.

the appropriate mask pattern at the end of the test phase. This is then loaded into the scan path registers before the configuration phase commences.

An important point about the modified scan path register is that it can be fully tested before being relied upon to peform the fault masking process. Both of the inputs and the output of the AND gate can be checked, as can the lower multiplexer and the latch. The parallel input of the upper multiplexer cannot be tested explicitly, but a fault in this area will be detected as a fault on the output of the circuit feeding the multiplexer.

9.8 Hardware Complexity of the Scan Path Approach

A transistor level circuit of the scan path cell shown in figure 9.8 contains 28 transistors. One scan-path cell is required for each of the three AVAILabilty

and REQuest outputs of the cell. We also need one scan path cell for each horizontal data line. The complexity of the scan path circuitry required per cell is therefore:

$$\text{Number of transistors} = 28(6 + L_s) \tag{9.4}$$

This means that the circuitry required for the scan path testing procedure is greater that that required for the control circuitry and at first sight this may not appear to be a sensible way forward. However, the testing strategy enables testing not only of the control circuitry, but also of the interconnections between processors and of the output stages of the self-test circuitry. The scan path design is also such that it can be fully tested from the external tester before being used to mask out faults. It is believed therefore that the overhead of the scan path circuitry is acceptable if it is essential that correct operation of the *WINNER* circuitry is to be *guaranteed.*

9.9 Other Tests

There are three areas which have not been tested by either the processor self-test procedure or the scan path test of the control circuitry. These are:

- the signature comparator circuitry,
- the vertical bypass circuitry used in the 1-dimensional *WINNER* algorithm,
- the interconnections between the control circuitry and the processor in the horizontal direction.

Tests to check these components are described in the following sections, and involve the scan path testing procedure described above.

9.9.1 Testing of the Signature Comparator

The self-test procedure should detect faults in the processor with a high degree of reliability, and the presence of the faults will show up when the test

signature is compared with the template signature. It is possible, however, for a fault to be present in the comparator itself, and to propagate an incorrect go/no-go indication to the control circuitry. There are many ways in which this problem can be overcome. We present two methods. The first is very simple and based on triple modular redundancy, but cannot guarantee to detect all faults. The second involves more additional circuitry but is capable of detecting any faults.

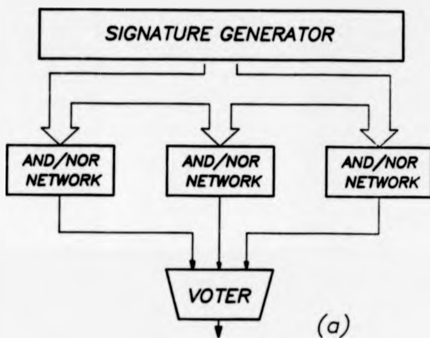
The TMR Approach

In this approach, the gates used in the comparator shown in figure 8.6 are triplicated, as shown in figure 9.9(a). A voting circuit is then used to deliver the output go/no-go signal to the control circuitry. The voting circuit required is very simple and is shown in figure 9.9(b). Because of its simplicity it should have a very high probability of correct operation, but there is always a small probability of a fault occurring in the voting circuit itself.

Fully Testable Comparator

There are many ways in which a scheme to enable full testing of the comparator circuit can be designed. In the scheme to be presented we attempt to minimise the amount of testing required at the expense of including a small amount of extra hardware and for this reason a serial method of comparison is used rather than the parallel method presented previously. The approach is illustrated in figure 9.10. The template signature is stored in a shift register which is entirely separate from the LFSR used in the compactor. Once the self-test has been completed, the signature and the template signature are clocked out of their respective registers and compared bit-by-bit in a single exclusive-OR gate whose output feeds a JK latch designed to remember any occurrences of a difference between the two signatures. The latch output then provides the go/no-go signal to the control circuitry.

The JK latch and the exclusive-OR gate can both be tested during the test of the control circuitry via the scan paths since the value of the go/no-



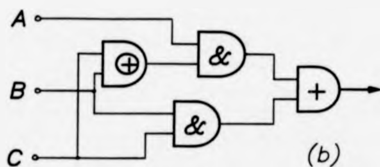


Figure 9.9: A TMR based signature comparator.

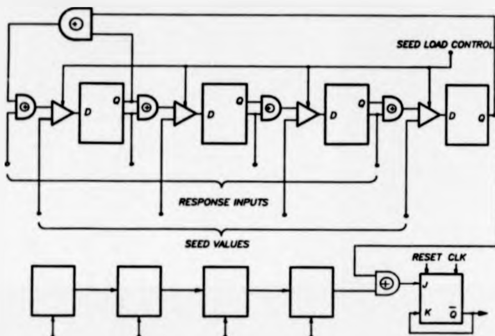


Figure 9.10: A fully testable signature comparator.

go signal on the JK latch output alters the function of the control circuitry. The two states of the latch can be exercised during testing in the following manner. A reset signal is required to switch the latch into the correct state (Q=0) prior to signature comparison and this can be used to check the *go* output value. More importantly, the *no-go* output state can be checked by using a seed in the LFSR whose lsb is different from that of the template signature. This will cause the exclusive-OR gate to detect an error and will set the latch into the no-go state (Q=1). In this way the user can be sure that the comparison circuit is operating correctly.

9.9.2 Vertical Bypass Circuitry

The vertical bypass circuitry is only used in the 1-dimensional *WINNER* algorithm, but it is an important component since all bypasses must be functional if the array is to operate correctly. Figure 9.11 illustrates a cell in which the vertical bypass circuitry can be tested in a simple manner.

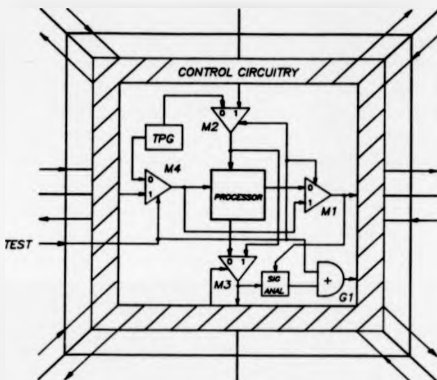


Figure 9.11: A fully verifiable *WINNER* cell.

The diagram shows the processor and control circuitry within the cell, together with the test pattern generator (TPG) and signature analyser, which together perform the processor self-test function. The vertical bypass function is performed by the multiplexer M3. An additional multiplexer M2 has been included to permit selection between the test pattern generator and the Northern data input to the cell. With M2 and M3 selecting the northern data input and the bypass line respectively, data will pass through the cell unchanged in the vertical direction. When cells are connected in an array, a complete column of cells can be checked for correct operation of the vertical bypass by ensuring that data applied at the top edge of the array emerges unchanged at the bottom edge. If any bypasses are found to be faulty, the entire array must be discarded. However, the circuitry involved is small, and the probability of achieving a fully functional array is high.

9.9.3 Horizontal Interconnections

The connections between the control circuits in adjacent cells are checked automatically during the scan path testing procedure. However, those between the control circuitry and the processor are only partially checked during the processor self-test procedure. These connections can be fully checked using the approach shown in figure 9.11 in which the multiplexers M1 and M4 have been included for the purpose. These multiplexers bypass the processor in the horizontal direction and are used in the bypass mode during the test of the control circuitry by the scan path procedure. Signals can then be passed through the cell and should be detected unchanged at the output. At the end of the test, M1 is controlled to select the data emerging from the processor, while M4 continues to select the horizontal input data from the control circuitry. All other parts of the circuitry are fully tested by the self test procedure.

9.10 Benefits of the Scan path Test procedure

The control circuitry test procedure provides a number of benefits which are listed below.

1. A small number of simple test patterns is required,
2. The number of distinct test patterns is independent of the size of the array, and of the function of the processor used in the array,
3. Interconnections between cells are checked automatically,
4. Faults in the control circuitry and cell interconnections can be masked, providing a second level of fault tolerance in the array,
5. The scan path circuitry be fully tested before being required to perform any role in which it must function correctly,

In addition, the vertical and horizontal interconnections between the control circuitry and the processor in each cell can be fully checked, as can the operation of the signature comparator.

9.11 Comments

In this chapter we have proposed two techniques which could be used to reduce the vulnerability of the *WINNER* control circuitry. The techniques are quite different and are likely to find use in different situations. The first technique involving duplicated control circuits requires less additional hardware than the external testing approach and is more suited to general applications, such as the fabrication of large integrated circuits, in which the configured array can be tested before use. For single faults occurring in each pair of control circuits, the duplication approach has the same performance as the external testing approach, being able to mask any fault. The second technique, requiring external testing of the control circuitry can be used if an absolute guarantee is required that the control circuitry, self-test signature comparator, and cell interconnections are functioning correctly. This assurance may be required in remotely sited applications, where testing the configured system may be impossible.

Chapter 10

WINNER Demonstrator

10.1 The Need for a Demonstrator

In previous chapters we have described the operation of the *WINNER* self-organising algorithm and its associated testing strategy, and illustrated the performance of the approach by simulation at both functional and hardware levels. This may be adequate as a purely academic investigation of the concept of self-organisation but is not sufficient if the work is to eventually be realised in a practical system. The *WINNER* techniques could be used in either a monolithic device such as a large area integrated circuit, or in a system required to have a high availability, which might be constructed from many separate modules, perhaps printed circuit boards containing components. For designers of these systems, the existence of a simple but practical demonstration of the *WINNER* algorithm would be a valuable aid to increasing confidence, understanding and appreciation of the benefits the approach can offer.

10.2 Type of Demonstrator

The most attractive approach to demonstrating the *WINNER* algorithm would be to apply *WINNER* to a large area integrated circuit which would normally be expected to have a yield close to zero, and to show that the use of *WINNER* in the device allows a yield significantly above zero to be

achieved. However, the design and fabrication of such a device would constitute a thesis in its own right and cannot be tackled within the constraints of the current project. Furthermore, a monolithic demonstrator would be very inflexible in the sense that the distribution of faults for a given device is fixed. This means that the ability to examine the response of the configuration algorithm to different fault distributions would be limited.

A more tenable approach, is to construct an array of printed circuit boards (pcbs) each comprising a single *WINNER* cell containing processor, control circuitry and test circuitry. This approach has been chosen for the *WINNER* demonstrator for this project and has several advantages over a monolithic device as a first demonstrator. Firstly, it can be completed within the project timescale, secondly, the cost involved in fabricating the demonstrator would be relatively low compared to a monolithic device, and thirdly, the larger physical scale of the demonstrator will present greater opportunity for producing a design whose operation can be understood by lay observer. This could be achieved by the inclusion of extra features such as lights to indicate the positions of faulty cells and configured rows, and the ability to manually introduce faults into the array to demonstrate operation of the configuration and fault masking processes.

10.3 Demonstrator Objectives

The objectives for the *WINNER* demonstrator are as follows:

1. To verify correct operation of the *WINNER* self-organising algorithm when implemented in hardware.

2. To provide a visual illustration of the operation of the algorithm that can be appreciated and understood by both expert and non-experts observers.

3. To verify the operation of the testing strategy and the procedure for masking control circuitry faults.

4. To act as a first prototype which could lead towards development of a large area silicon demonstrator based on *WINNER*.

10.4 Demonstrator Specification

Consideration of the hardware implementation of *WINNER* has led to the following specification for the demonstrator.

1. The demonstrator should be capable of configuring a functional array with 4 rows and 4 columns from an array containing 6 rows and 4 columns. It should use the one-dimensional *WINNER* algorithm since as described in chapter 6, this is the algorithm most likely to be used in practice.

2. The processor design is to be kept simple by avoiding the use of self-test circuitry since this is the subject of much research elsewhere and is known to be possible. It is considered adequate for the purposes of this demonstration to use a switch on each cell to indicate the condition of the processor. This provides the flexibility to alter the fault distribution to test and illustrate various array configurations.

3. The configuration of the functional rows should be indicated visually using LEDs or similar illuminating device. It should also be possible to configure the array in *slow motion* so that the interaction between REQuest and AVAILability signals can be observed.

4. Correct functioning of the configured processor array must be demonstrated and it should be obvious to an observer that this is the case. This could be achieved by allowing the array to perform its function on known input data and observing the output.

5. The necessary row selection circuitry for both inputs and outputs should be included.

6. The modified scan-path testing procedure is to be included and should permit detection and masking of real faults introduced into the control circuitry and inter-cell connections, for example, by shorting wires together. This will necessitate the development of an external tester for generating test patterns, evaluating responses from the array and producing the appropriate fault masking pattern. The location of the cell in which the fault was detected should be indicated visually.

10.5 Processor Array Function

Since the main purpose of constructing a demonstrator is to prove the operation of the *WINNER* algorithm and its associated test strategy, the function of the processors used in each *WINNER* cell and the function of the processor array itself are of secondary importance. For this reason an array of very simple processing elements (PEs) has been chosen. The array performs the multiplication of two binary numbers A and B using ripple-through PEs and is shown in figure 10.1(a). The ith bit of A, a_i, $0 \leq i \leq 3$ is broadcast to each PE in the ith row, while the ith bit of B, b_i, $0 \leq i \leq 3$ is broadcast to each PE in the ith column of the array. Each PE takes inputs a_i and b_i, s and c and generates a new value for s and c as shown in figure 10.1(b). The values of s and c are formed by multiplying the incoming bit from A with that from B and adding the product to the incoming s and c values. The main array of PEs in figure 10.1(a) generates all the partial products required for the final result and also performs part of the task of summing the partial products. The remainder of the task is carried out by the extra row of full adders shown at the bottom of the main array. The product of A and B emerges as shown. The multiply function carried out by this array is ideal for this project because the PEs are very simple and, in addition, an observer will instantly recognise whether or not the array is producing the correct product.

It will be noticed that the array shown in figure 10.1(a) contains diagonal

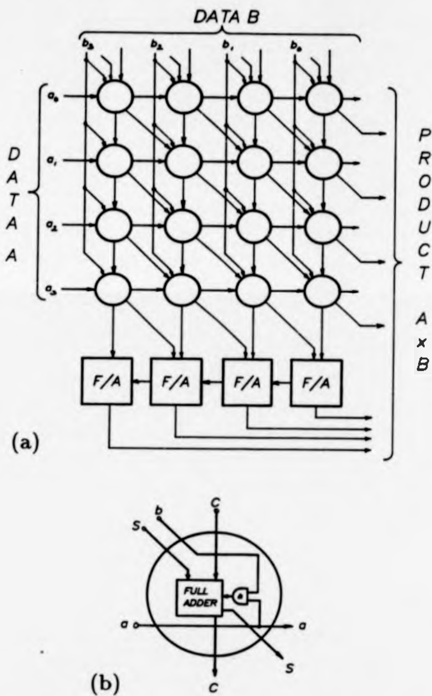


Figure 10.1: 4-bit by 4-bit array multiplier: (a) Multplier circuit, (b) Single cell.

connections between PEs. These must be removed before the *WINNER* algorithm can be applied. Figure 10.2(a) shows the same array redrawn without diagonal interconnections. An extra dummy connection has been made as shown in figure 10.2(b) which enables the diagonal path to be generated from two orthogonal paths. The function of the PE is otherwise unchanged.

10.6 Implementation Options

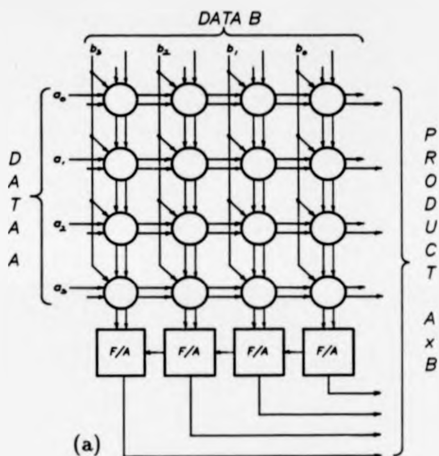
The circuitry for the cell to be used in the demonstrator can be implemented in a number of ways. Those which have been considered are described below and one of the approaches is selected for the design.

10.6.1 LSI Components

The most straightforward implementation approach is to use standard, off-the-shelf components such as those available from the TTL or CMOS families of logic circuits. The appropriate components are simply selected from the catalogue and the pcb is designed to interconnect them in accordance with the circuit diagram. However, although the circuitry required in the cell is relatively simple, 17 LSI chips would be required which together with the necessary LEDs etc would result in a pcb of about 5 inches square. It was felt that this would in turn result in an unacceptably large array size.

10.6.2 Custom Chip or Gate Array

In this approach, the circuitry for an entire cell would be placed on a single chip. A cell library is used to provide standard functional blocks such as gates and latches and these are placed on the silicon and interconnected as required. The completed chip could then be used on a pcb together with the appropriate LEDs and switches. This approach would incur the least effort in pcb design but would be relatively inflexible in terms of ability to introduce faults at various locations within the cell.



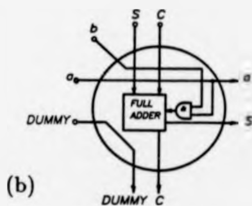


Figure 10.2: Multiplier array redrawn with orthogonal interconnections: (a) Array, (b) Single cell.

10.6.3 EPROM Implementation

Since much of the circuitry required in the *WINNER* cell is combinational logic, it is possible to design a compact circuit using EPROMs. EPROMs are Programmable Read Only Memories, which can also be erased for alteration or re-use by prolonged exposure to ultra-violet light. EPROMs can implement a truth table of combinational logic if data corresponding to the output required from each input address is stored in the memory. EPROMs are primarily designed for use in computer systems and as a result store 8-bit data words. One 8-bit word is therefore output by the EPROM for each address input. The number of bits in the input address is determined by the size of the EPROM, with for example, a 1K x 8 EPROM having a 10-bit address. The most cost effective EPROM at the time the design of the demonstrator was being undertaken was an 8k x 8 device, having 13 address inputs. The advantage of using EPROMs in the demonstrator is that the number of chip packages can be reduced. This reduces the pcb size and design time. However, the cost of using EPROMs is approximately the same as that for LSI devices.

EPROMs have some of the advantages of both the custom chip approach and the LSI approach for this project. For this reason they have been chosen as the method for implementing the combinational logic required in the cells. Careful partitioning of the circuit will be necessary to minimise the number of EPROMs required.

10.7 Demonstrator Circuitry Requirements

A block diagram of the demonstrator is shown in figure 10.3, and shows the main components of the system. In addition to the 6-row by 4-column *WINNER* array, there are other circuits which are required to perform functions such as generation and display of input data and results, row selection for the input and output of data to the array, and interfacing to the BBC computer. The circuits for these functions are described in more detail in the following

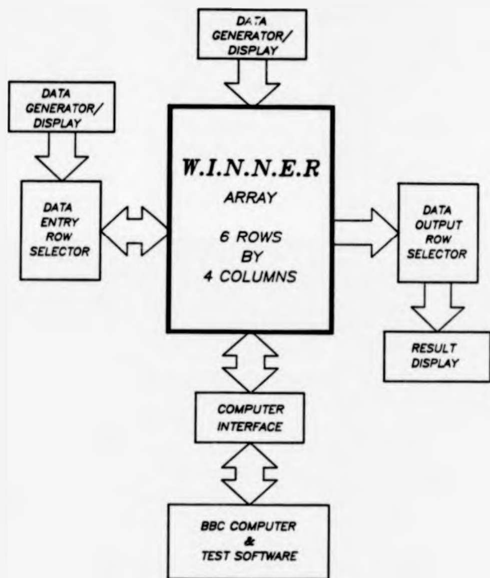


Figure 10.3: Block diagram of the *WINNER* demonstrator.

sections.

10.7.1 Circuit for the *WINNER* Cell

The circuitry required to implement a cell to the to the specification of the demonstrator is illustrated in figure 10.4. The functions shown in blocks are implemented in separate EPROMs and are shown in detail in figures 10.5, 10.6 and 10.7. The inputs and outputs on the right hand side of the cell each pass through a latch which is additional to the latch required for the scan path testing scheme. The extra latches serve no function in the *WINNER* algorithm, but allow the array to be configured one step at a time by clocking the latches slowly. The partitioning into circuits suitable for EPROM implementation has been carried out so that the *WINNER* control circuitry which generates REQuest and AVAILability signals is in one EPROM, while the processor and data routing circuitry is in another. The logic involved in the scan path testing circuitry requires two EPROMs for its implementation as shown.

The logic levels of the input and output REQuest and AVAILability signals are indicated by LEDs which are illuminated when the signal is at a high logic level. These LEDs are placed on the periphery of the cell corresponding approximately to the positions where the signals enter or leave the board. A switch is included on each cell to simulate the output of a self-test circuit. The switch controls an input to the EPROM which contains the control circuitry. A green LED at the centre of the pcb is used to indicate the result of the simulated self-test; if illuminated, it indicates that the processor is functional, and vice-versa. Another LED, also in the centre of the pcb is used to indicate the position of cells which have been found to contain faults during the scan path testing of the cell. A cell found to have a fault will cause its (red) LED to flash. Test points have been included on each cell to allow the introduction of faults on the REQuest and AVAILabilty signal lines. These are not the only faults which can be applied, but they are the ones most likely to be used to demonstrate the fault masking process.

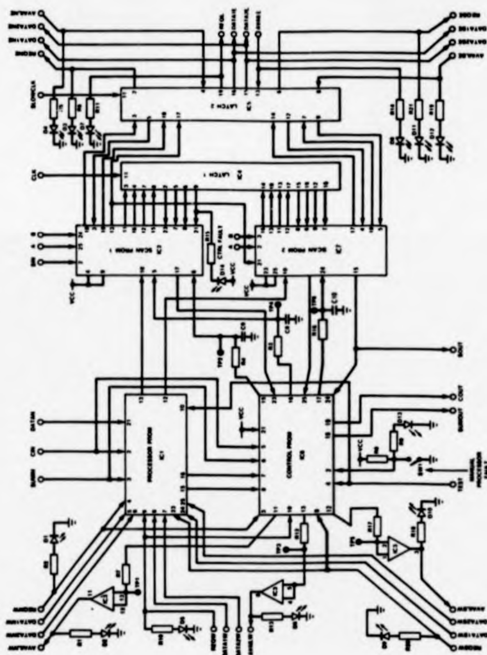


Figure 10.4: Circuit diagram of the cell used in the *WINNER* demonstrator.

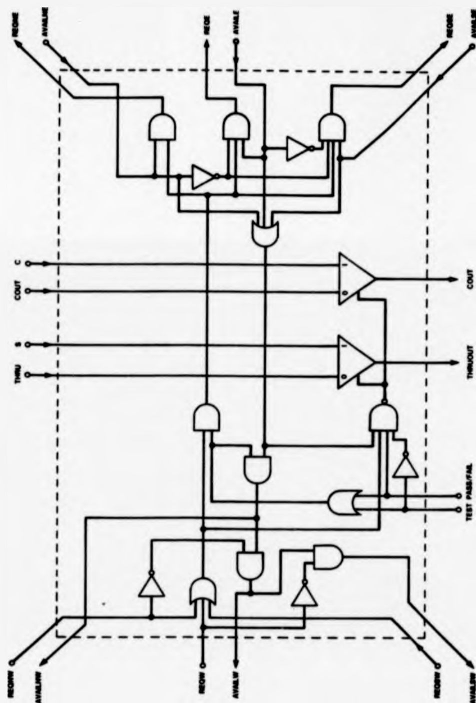


Figure 10.5: Equivalent circuit of the Control PROM.

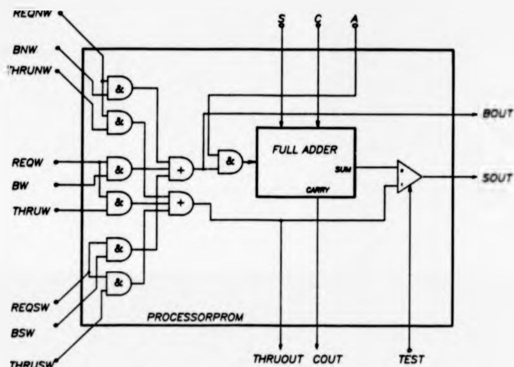


Figure 10.6: Equivalent circuit of the Processor PROM.

10.7.2 Other Circuitry

Extra Row of Full Adders

This circuit has been implemented in a single EPROM, and takes its inputs from the bottom of the *WINNER* array.

Input/Output Row Selection Circuitry

The circuitry used in the input and output row selection arrays is purely combinational and can therefore be implemented using EPROMs. The circuits of the cells of the input and output selection arrays together with the circuitry placed on each EPROM are shown in figures 10.8 and 10.9 respectively. Sufficient circuitry for two rows of cells has been placed in a single EPROM, and three EPROMS can be cascaded to form input and output arrays of the required dimensions.

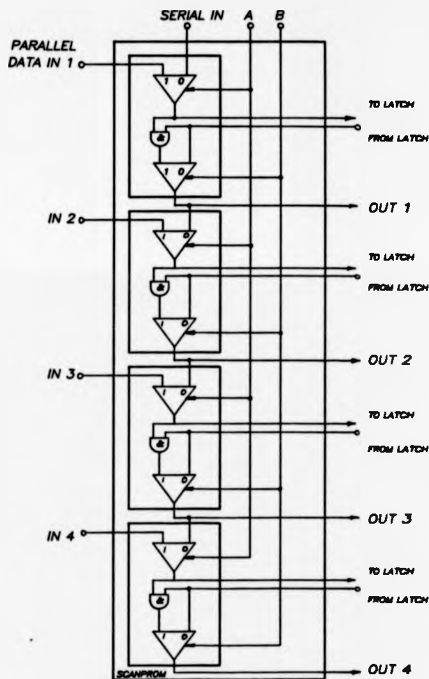


Figure 10.7: Equivalent circuit of the Scan PROM.

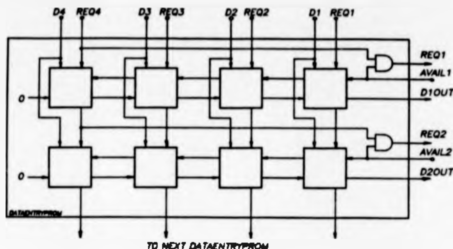


Figure 10.8: Equivalent circuit of the data entry PROM.

Data Generation and Display

The two data words to be multiplied together on the configured array are each generated by a circuit comprising a 4-bit binary up/down counter circuit clocked by debounced switches. The value in the counters is displayed on 7-segment LED displays with one of the values being fed to the input data row selection circuit, and the other directly to the cells in the array. The display circuitry also receives the product generated by the array (via the output data row selection circuitry), and displays the result. This should therefore be equal to the product of the two input numbers when the array has been configured.

Scan Path Testing Circuitry

Most of the circuitry required to implement the scan path testing procedure is embodied within the cells themselves. However, since a scan path is required on the left and right hand sides of each cell and the cells have been designed with a scan path on only the right hand side, an extra scan path is required on the far left hand side of the array. This has been implemented in the same manner as those within cells, except that hand wired boards have been

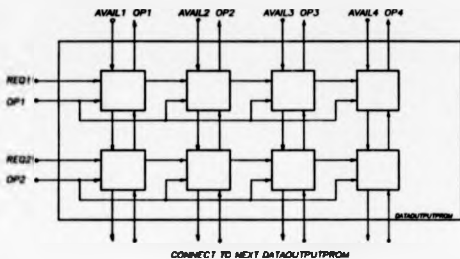


Figure 10.9: Equivalent circuit of the data output PROM.

used.

10.8 External Testing

The scan-path testing procedure requires the use of an external tester which is known to be fully functional. The tester must be capable of applying test patterns to the array via the scan paths, receiving the responses from the array, and generating the appropriate fault masking patterns. These must then be loaded into the scan paths before configuration commences. It is possible to design a piece of dedicated hardware to perform this task but it would be very inflexible to changes in test pattern, and would take considerable time to design and construct. A better approach, and the one adopted in this project is to use a computer and program it to perform the required task. We have used a BBC computer, with the I/O and printer ports being used to transfer data. A circuit has been designed which provides the necessary interface between the computer and the array.

10.8.1 Computer Interface circuitry

The BBC computer has insufficient input/output ports to allow one port to be dedicated to each of the inputs and outputs which require to be monitored. For this reason an interface circuit between the computer and the array is required. The function of the interface circuit is to provide a set of latches with one latch for each input and output. A multiplexer arrangement is included so that different subsets of latches can be selected. The computer can then either write to the subset, or read from it as required.

10.8.2 Testing Software

The BASIC program which runs the test sequence is included in Appendix C together with a brief description of its operation. The program makes full use of the PROCedures available in BBC BASIC and is therefore relatively easy to read. The instructions for setting the various array inputs to TRUE or FALSE have also been written so that a simple procedure call is all that is required to change the value of an input. The program has been designed so that it operates on a menu basis. The operation of the array can therefore be controlled without knowing the details of how the program operates.

10.9 Demonstrator Results

A photograph of the completed demonstrator system together with computer and computer interface is shown in figure 10.10. A close-up photograph of a single cell of the *WINNER* array is shown in figure 10.11, and a configured array is shown in figure 10.12. This has been taken through a red filter so that only the REQuest signals can be seen. These indicate the positions of the functional rows.

The completed demonstrator operates fully to specification and has been demonstrated to many people. The configuration can be observed in slow motion and the interaction between REQuest and AVAILability signals can be seen clearly. The display of the product generated by the array is always

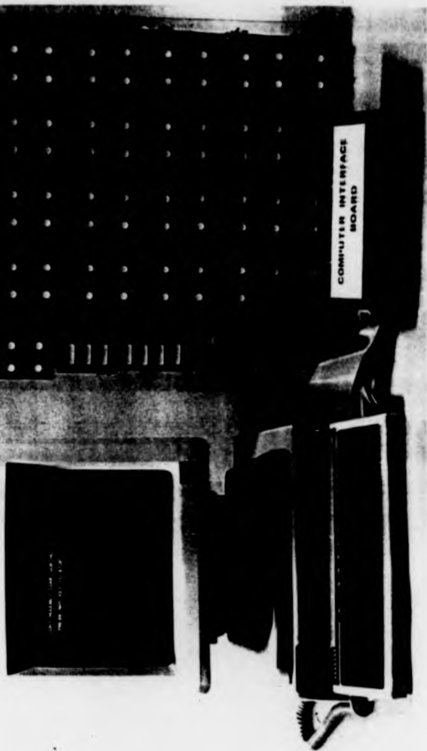


Figure 10.10: Photograph of the demonstrator showing the *WINNER* array, computer interface and computer.

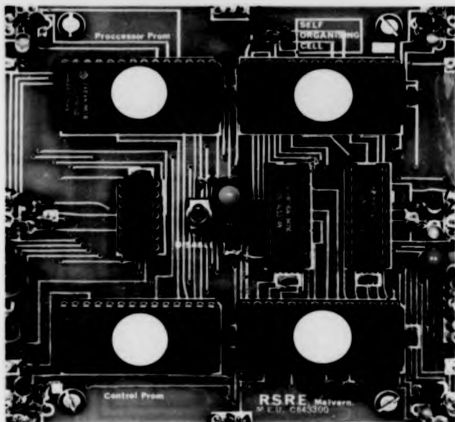


Figure 10.11: Single cell of the demonstrator array. The values of the REQuest and AVAILability inputs and outputs are indicated by the red and green LEDs respectively.

correct when the scan path test has been carried out and when the array is in configuration mode. When the distribution of processor faults is altered by adjusting the positions of the switches, the display of the result generated by the array multiplier returns to the correct value after a brief transient period while the array is reconfiguring.

The testing procedure based on the BBC computer is rather slow. However, this is a feature of the BBC and the way in which the program has been written. It could be greatly speeded up by writing critical portions of the program in machine code or by using a more powerful computer. A dedicated circuit to generate the test patterns would also operate much more rapidly.

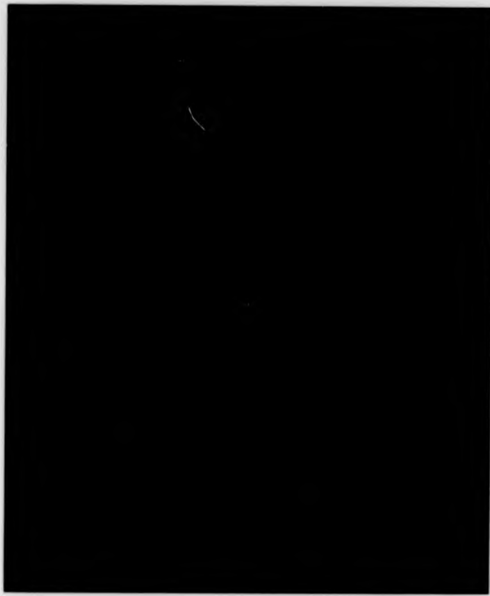


Figure 10.12: Photograph of the demonstrator array when configured for a particular fault distribution. (Taken through a red filter to highlight the configuration pattern.)

The problem is of little consequence to the *WINNER* algorithm itself.

Real faults, such as stuck-at-1 or 0 can be introduced into the control circuitry via the test points on the pcbs, or by other means, and these are all correctly detected by the test sequence and the culprit cell identified by a flashing LED. Subsequent configuration of the array shows that the cell into which the fault has been introduced has been correctly avoided and is not part of a functional row even though it may appear to be outputting TRUE AVAILability signals. The resulting product is also correct.

10.10 Concluding Remarks

The main purpose in constructing a self-organising array based on *WINNER* was to demonstrate that the ideas proposed in chapters 5 and 9 are sound and practically realisable. We have shown this to be the case, and have not encountered any unforseen problems. Further work towards the development of a large area monolithic device based on *WINNER* can now be undertaken, confident in the knowledge that the underlying configuration and fault-masking techniques are sound.

Chapter 11

Applications of WINNER

11.1 Introduction

The *WINNER* self-organising technique described in the previous chapters can potentially find use in several different applications. In this chapter we briefly outline some of these applications and provide an indication of the specific details of each in terms of special implementation requirements. The first section is concerned with the types of processor array which could benefit from a technique like *WINNER*, while later sections focus on the application of *WINNER* to various different implementation approaches. The chapter is not meant to provide complete details of how *WINNER* would actually be used in each application, since this would constitute a thesis in its own right.

11.2 Application to Processor Arrays

As was described in chapter 2, arrays of processing elements are finding increasing use in high performance computers and in digital signal processing applications. In this section we briefly mention several processor arrays which are currently of interest in system design and in which the *WINNER* self-organising technique could be used.

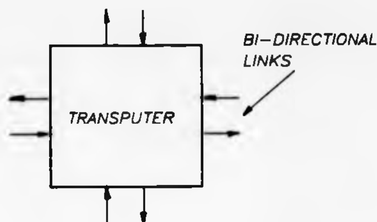


Figure 11.1: Schematic view of the Transputer.

11.2.1 The Transputer

Transputer chips are manufactured by Inmos Ltd, see Inmos (1984). A Transputer is essentially a RISC microprocessor with additional circuitry to provide it with four high speed bi-directional serial data links. The links are completely asynchronous and have been designed to allow easy interconnection of many transputers in an array. A schematic view of a Transputer is shown in figure 11.1. The connectivity of the array can be designed to suit the problem in hand. For orthogonally interconnected arrays, or arrays which can be transformed so that they contain only orthogonal interconnections, it would be possible to use *WINNER* to provide the basis of a fault tolerant capability. Such arrays are useful in image processing applications, where part of an image might typically be allocated to each processor in the array. An array of transputers is a SIMD machine since each transputer can contain a separate program and operate on separate data.

An advantage of the transputer in the context of *WINNER* is that the serial nature of the interconnecting links means that only a pair of wires requires to communicate in each of the four directions. This means that routing of the data lines can be achieved with very little hardware. Currently,

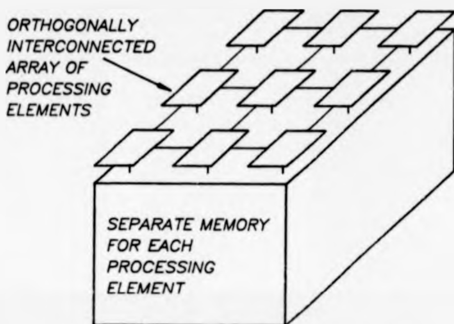


Figure 11.2: Schematic view of the ICL DAP.

the transputer is not able to test itself, so some external mechanism would be required to perform this function.

Being a single chip, the transputer also lends itself to wafer scale integration, although at present, the size of the chip means that the necessary yield of chips on the wafer is likely to be insufficient for economic production.

11.2.2 The Distributed Array Processor (DAP)

The DAP (Reddaway, 1973) is built by ICL and is a high-performance general purpose array processor which supports a high level language called DAP Fortran. The DAP can be programmed to perform many tasks in the field of digital signal processing including speech processing and image processing. It is a SIMD machine and contains a 64 by 64 array of bit-level processing elements each having 4kbits of memory. The processing elements communicate with neighbouring elements via an orthogonally interconnected mesh as shown in figure 11.2. Each processing element in the DAP is currently implemented using several separate chips but communication between elements

is by single bit signal lines, which is ideal for *WINNER*.

11.2.3 Digital Signal Processing Arrays

Under this heading is grouped a number of arrays which essentially perform a fixed task although they are programmable in terms of coefficients and operands. Their purpose is to perform a limited class of operations very rapidly on streams of high speed data. Functions may include matrix operations, filtering operations such as filter banks as well a predefined operations on images, such as edge detection and segmentation. Arrays can be built from standard chips such as multipliers, cascadable filters or microprocessors specifically designed for use in digital signal processing arrays, such as the Systolic Node chip (Hargrave, 1986) which is currently under development at STL.

Since the processing elements tend to be available as single chips there is great potential for improved performance by implementing arrays as monolithic circuits.

11.3 Application in Array Implementation

The following applications of the *WINNER* self-organising technique are presented in order of increasing difficulty of implementation.

- High availability applications,
- Implementation as part of an *active substrate* in a silicon hybrid,
- As the configuration technique in Wafer Scale Integration,

It should be possible to use *WINNER* in all of the above applications without much extra effort and without any modification to the basic *WINNER* algorithms or hardware. In the following sections, we describe the essential features of each application and the main problems which must be addressed.

11.4 High Availability Systems

A ***High Availability*** system is one which although not able to tolerate faults as they occur can nevertheless rapidly reconfigure itself to avoid the fault. This means that the user of the system would receive corrupted output data for a short period while the system reconfigures. High availablity systems are useful both in remote applications such as satellites, where manual repair is not practical, and in applications where the time to carry out manual repair would be unacceptable.

With the increasing interest in parallel processing techniques as a means of achieving high performance computers, there are several computing machines available which consist of two-dimensional arrays of processing elements. In general, currently available technology does not permit these arrays to be implemented as single chips due to the overall complexity of the system, and the arrays are usually partitioned into a number of printed circuit boards. When the processor array is manufactured, it is tested to check that it is fully functional. However, when in service, it is possible for components or interconnections to fail and automatic repair would be advantageous in many applications.

WINNER could be used to provide this automatic repair capability. A special feature of this application is that all the circuitry involved, including that specific to ***WINNER*** is fully functional at the start of the life of the system. This is different to the other potential applications of the technique. As a result, a straightforward implementation of the ***WINNER*** control circuitry together with self-test of the processor is all that is required.

11.4.1 Special Considerations

Self-Test

Some mechanism is required by which the array can be informed that it contains a fault. The simplest, but not very satisfactory approach could involve the user who could initiate a retest and reconfiguration when he

detects an error. A better method would be to automatically perform a self-test at regular intervals or whenever the processor is idle. Any detected faults could then be avoided by reconfiguration.

In systems constructed from a number of interconnected printed circuit boards, the most likely failure mechanism is a fault occurring at the interface between different levels of integration. In a typical system inter-chip communication might include the following interfaces: chip-to-package, package-to-board, board-to-backplane, backplane-to-board, board-to-package, and package-to-chip. Since the interconnections between individual processors are likely to involve communication across some of these interfaces, the self-test approach used in the system must be capable of checking the inter-processor connections.

Control Circuitry

At the start of the life of the system, each control circuit will be fully functional. If the control circuitry is implemented on a single chip it will have a small probability of failure compared with the rest of the processors. However, since the most likely cause of failure is the interconnections between boards, the operation of the configuration circuitry could be significantly affected since signals are passed between processing elements. Faults in the interconnections between control circuits in adjacent cells can be tolerated by taking advantage of the *active* and *passive* conditions which were described in chapter 9. In this application, we simply need to design the input lines to the control circuits such that if the line is open-circuit, it is pulled low and as a result appears to be a passive input level. This will result in the circuit receiving the passive signal treating its neighbour as if it was a faulty cell, and ignoring it.

11.4.2 General Comments

One of the advantages of using *WINNER* in a high-availability application based on an array of pcbs is that only a small number of redundant elements

are required. This is because the manufacturer can ensure that all of the array elements are fully functional before installing the system, and sufficient spare capacity need only be provided to cope with the expected failure and repair rates. The minimum requirement is one spare row, but provided that not more than one fault occurs in any column, several faults can be tolerated by the spare cells. This results in a very cost-effective solution and enables the array to be configured automatically whenever a fault is detected.

11.5 Silicon Hybrids

In a conventional hybrid, individual dice, or chips mounted in leadless chip carriers are bonded to a ceramic substrate. The substrate provides the required interconnections between the chips through metal tracks (usually gold) which are screen-printed onto its surface. This approach has been used for many years and offers much reduced size of system implementation. However, further miniaturisation using the conventional hybrid technique is limited by the low density of interconnect possible on the ceramic substrate. For this reason the replacement of the ceramic substrate with a silicon wafer or part wafer has been the subject of research for several years.

A Silicon Hybrid is a compromise between a conventional hybrid circuit and Wafer Scale Integration and has many advantages over conventional hybrids, a few of which are listed below: (Hagge, 1988)

- Wafers can use integrated circuit lithography techniques to generate the high density interconnections,
- Silicon hybrids can have *active substrates*, in which some components are fabricated in the substrate material and others bonded to the surface,
- The higher density interconnect means that chip-to-chip propagation delays are reduced due to shorter interconnections,

11.5.1 Requirement for Fault Tolerance

An advantage of both the conventional hybrid approach and the silicon hybrid approach from the point of view of fabrication is that each chip to be bonded to the substrate can be fully tested before use. In addition, the interconnections on the silicon substrate can be checked by a probe test. During assembly, however, the bonding process may not have a 100% yield, which will result in some substrates being faulty. At present, the technology of silicon hybrids is immature and no firm details are available on the reliability of the bonding process. It is likely that the faulty bonds may be able to be repaired by being *reflowed*, but in applications involving large numbers of chips each with a dense pinout structure it is likely that repair may prove costly.

An approach which could avoid having to locate and repair bonding defects if a 2-dimensional array is to be fabricated is to use the *WINNER* technique and implement the control circuitry in the active substrate of the hybrid. A further advantage of this approach would be that faults occurring in-service can also be automatically tolerated.

Placing the configuration control circuitry in the active substrate means that it remains entirely separate from the chips being attached to the surface of the silicon. As a result, no alterations are required to the chips themselves and so standard components can be used. The only requirements are that the processors are self-testing and that the yield of the control circuitry is sufficient to allow a high percentage of the active substrates to be fully functional. There is already a trend towards self-testing chips since they reduce test times in conventional stand-alone chips and if this trend continues, such chips would be ideal for use in silicon hybrids.

11.6 Wafer Scale Integration

Wafer Scale Integration (WSI) is possibly the most demanding application of both the *WINNER* technique and the silicon process being used. In WSI, no

component can be guaranteed to work and the best one can do is to ensure that the process has a yield which, on average, produces acceptable results.

The original idea behind *WINNER* was to develop a configuration technique suitable for enabling WSI, or at least, large chips to become a reality for 2-dimensional arrays of processors. We believe that the techniques have been developed to the stage where it is appropriate to consider building a WSI demonstration. However, the configuration of the processors is not the only problem involved in WSI, and the following list highlights some of the other areas which must be considered to be successful in WSI.

- Control circuit yield,
- Power supply distribution and integrity,
- Clock line distribution and integrity,
- Disconnection of faulty processors,
- Fault model for the process being used,
- Lithograpic techniques for large area exposure,
- Packaging techniques,
- Thermal management within the package,

Some of these problems have been addressed within this project and the results reported in this thesis. These include techniques for improving control circuit yield, and a study of integrated circuit fault distributions. It is not possible to address the other problems here since they are beyond the scope of this thesis. However, some work has been done in the context of specific WSI architectures on the distribution of power and clock signals by Warren et al, (1986) and Coleman and Lea (1986). The problem of disconnecting faulty processors is discussed by Warren and Lea, (1987). Problems associated with packaging and thermal management have been considered by Pitt (1987).

Chapter 12

Conclusions and Further Work

12.1 Conclusions

In this thesis we have presented the results of a project concerned with research into configuration techniques suitable for use in Wafer Scale Integration. Motivation for the research has been generated from two independent directions. Firstly, the increasing trend towards regular parallel processing architectures is likely to have a significant, beneficial impact on the design of integrated circuits and enable very large chips to be designed with moderate effort. This increase in the level of integration should then feed back into the field of parallel processing in the form of reduced hardware costs etc. Secondly, these large chips are at present impossible to produce successfully due to the fabrication defects which are always introduced during silicon processing. A study of parallel processing trends and of the fault distributions occurring in integrated circuits has been carried out and the results have been presented.

A detailed study of published approaches to the problem of configuring a 2-dimensional array has been carried out. Each author proposes his own approach to arranging the switching elements which perform the routing of data to avoid fault elements. Many different methods of implementing the switching elements are also proposed and we have developed classifications for the switch organisation methods and implementation methods. We show that there is an alternative, novel approach to configuration which has not

been addressed in the literature, namely that of *Self-organisation*.

In the remainder of the thesis we have concentrated on developing the concept of self-organisation. We have proposed a novel algorithm, called *WINNER* by which a 2-dimensional array can avoid the faulty processing elements and organise itself into a functional array without any external assistance. We have performed simulations which compare the performance of *WINNER* with other published techniques. We show that the configuration performance of *WINNER* is comparable to other approaches when processor yield is 80% or more, although it becomes less competitive at lower yields. In addition, *WINNER* has several advantages.

- The array can configure itself automatically without the need for external assistance,
- The self-organising algorithm is fully convergent and cannot become unstable,
- The control circuitry associated with each cell in the array is simple, about 20 gates, resulting in a low hardware overhead. In fact for arrays with more than four spare rows, the overhead is less than the competing approaches,
- The control circuitry has the inherent ability to tolerate certain faults occurring within it. In addition, it can be designed so that it can tolerate a range of other faults without affecting the operation of the remainder of the array,
- The technique results in good utilisation of the functional processors particularly when processor yield is greater than 80%,
- No global control lines are required,
- The array is potentially capable of being *re*-configured in the event of an in-service failure and could therefore be useful for remotely sited equipment, or in equipment requiring a very fast repair time.

The fact that the hardware overhead of the *WINNER* technique compares favourably with competing techniques is a somewhat surprising result. It is due to the fact that the required configuration of the functional array is calculated by the control circuitry itself and as a result there is no need to use latches to store switch control information within the array.

A weakness of any fault tolerant strategy is that faults occurring in the switching circuitry can render the entire array unusable. This problem does not appear to have been addressed in the configuration techniques published in the literature. However, we have shown that the *WINNER* approach has the intrinsic ability to tolerate some faults in the control circuitry. In addition, we have developed a technique which enables many more control circuit faults to be tolerated automatically. We have also shown that all of the control circuits in the array can be fully checked by a simple external test if a simple novel scan-path arrangement is incorporated into the array. Faulty control circuits can subsequently be *masked* from the rest of the array before configuration takes place.

We have shown that the *WINNER* concept is practically realisable by constructing a small demonstrator based on an array of printed circuit boards. The array demonstrates the operation of the basic *WINNER* configuration algorithm and also allows real faults to be introduced into the control circuitry (by shorting signal lines high or low). These faults are then detected by the scan-path testing scheme and after their effect has been masked from the rest of the array correct configuration of the array is demonstrated. As described in the following section, the *WINNER* technique is currently being used by a UK company as part of their research into Wafer Scale Integration.

Finally, we have briefly outlined some of the applications which could benefit from the *WINNER* approach. These include high-availability systems, silicon hybrid construction and reliability improvement, and Wafer Scale Integration. We have identified several systems using orthogonally interconnected arrays of processors which could potentially benefit from the *WINNER* technique.

12.2 Suggestions for Further Work

As far as possible, the problems concerning the application of *WINNER* to orthogonally interconnected two-dimensional arrays has been addressed within this thesis. In the context of further research outside the scope of this thesis, however, there are several areas on which attention could be focussed.

12.2.1 Fabrication of a Monolithic circuit

The real test of *WINNER* will occur when it is used in the design and fabrication of a monolithic circuit. It is only when used in a real application that the technique will be fully tested and any weaknesses uncovered.

In the past few months, a British company has shown interest in the *WINNER* technique and intends to use it as the basis for its in-house research program into wafer scale integration. This is very encouraging both from the point of view of *WINNER* and of the long term view and commitment being shown by the company concerned.

The project will contain a number of stages, the first of which will be to demonstrate that the control circuitry can be fabricated with a sufficiently high yield and that some functional devices can be obtained. Initially only a very simple processing element will be used, such as that used in the demonstrator reported in chapter 10. If necessary, the dual-rail techniques described in chapter 9 will be used to increase control circuit yield. When it has been demonstrated that sufficient control circuit yield can be achieved, a larger processor will be used and a practical wafer scale circuit will be designed.

At present the project is at an early stage and no results have yet been obtained.

12.2.2 Extension to Tree Structures

It may be possible to apply the concept of self-organisation to other interconnection patterns, in particular to tree structures. Tree structures are of

interest in the field of artificial intelligence because they allow searching of rule bases to be carried out in parallel (Hillis, 1986). One of the attractions of the tree structure, particularly in wafer scale integration is that only a single processing element needs to communicate with the outside world. However, one of the problems of developing a self-organising tree structure is that the optimum configuration is likely to depend on the size of the tree which can be configured, and of course this will not be known at the start of the configuration process.

12.2.3 Fault Tolerant Switching Networks

There is much interest in the literature in *Interconnection Networks* for computers; see for example the review by Adams, Agrawal and Siegel (1987). Engineers are interested in designing switching networks capable of interconnecting any of a group of machines to any other in the group. This represents a challenge in itself in terms of designing a switch with the appropriate trade-off between hardware used to implement the switch and the flexibility of the switch. Also of interest is the ability of the interconnection networks to tolerate faults, so that in the event of a fault in the switch, an alternative route between the required machines is available.

The link between this work and *WINNER* is that *WINNER* is a type of fault tolerant switching network. It may be possible, with some modification of the particular *WINNER* algorithm presented in this thesis, to apply a similar approach to this area. For example, in an interconnection network, the removal of a connection between two parties should have no effect on the machines still connected. In the basic *WINNER* approach, the removal of a functional row would cause all functional rows below the one removed to reconfigure to take up the vacated space.

An initial, very brief study has been carried out which suggests that this problem can be overcome by including latches in the *WINNER* control circuitry to store the configuration pattern of each functional row until the row is no longer required. Space vacated by the removal of interconnections would

be used wherever possible when new interconnection are set up. However, much more work is required in this area before useful algorithms emerge.

Bibliography

Adams G B, Agrawal D P and Siegel H J (1987), *Fault-Tolerant Multistage Interconnection Networks*, IEEE Computer, June 1987, pp 14-27.

Aubusson R C and Catt I (1978), *Wafer Scale Integration - a Fault Tolerant Procedure*, IEEE J of Solid State Circuits, Vol 13, No 3, pp 339-344.

Avizienis A, (1976), *Fault-Tolerant Systems*, IEEE Trans. on Computers, C-25, No.12, December 1976, pp 1304-1312.

Barsuhn H (1969), *Functional Wafer - A New Step in LSI*, 1977 European Solid State Circuits Conference (Ulm, Germany), pp 79-80.

Bentley L and Jesshope C R (1986), *The Implementation of a Two-dimensional Redundancy Scheme in a Wafer Scale High Speed Disk Memory*, in *Wafer Scale Integration*, C R Jesshope and W R Moore eds, Bristol UK, Adam Hilger, pp 187-197.

Calhoun D F (1969), *The Pad Relocation Technique for Interconnecting LSI Arrays of Imperfect Yield*, Proc 1969 Fall Joint Computer Conference, pp 99-109.

Catt I (1981), *Wafer Scale Integration*, Wireless World, July 1981, pp 57-59.

Chapman (1985), *Laser-Linking Technology for RVLSI*, Proc International Workshop on Wafer Scale Integration, Southampton University, July 1985, pp 204-215.

Chen W, Mavor J, Denyer P B and Renshaw D (1988), *Superchip Architecture for Implementing Large Integrated Systems*, Proc IEE Part E, Vol 135, No 3, May 1988, pp 137-150.

Cliff R A and Rao T R N (1974), *Improving Yield of LSI Memory Chips by the Application of Coding*, 8th Princeton Conf on Information Science and Systems, pp 386-390.

Cole B C (1985), *Wafer Scale Integration Faces Pessimism*, Electronics Week, April 1, 1985, pp 49-53.

Coleman J N and Lea R M, (1986), *Clock Distribution Techniques for Wafer Scale Integration*, in *Wafer Scale Integration*, eds Moore and Jesshope, Adam Hilger, 1986, pp 46-53.

Daniels R G and Bruce W C (1985), *Built-in Self-test Trends in Motorola Microprocessors*, IEEE Design and Test, April 1985, pp 64-71.

Eckert J P Jr, Weiner J R, Welsh H F and Mitchell H F, (1951), *The UNIVAC System* in Bell and Newell 1971, pp 157-169.

Eichelberger E B and Williams T W, (1977), *A Logic Design Structure for LSI Testability*, 14th Annual Design Automation Conference, June 1977, pp 462-468.

Evans R A, McCanny J V, McWhirter J G, McCabe, Wood D and Wood K W (1983), *A CMOS Implementation of a Systolic Multi-bit Convolver Chip*, Proc VLSI-83, Trondheim, Norway, 1983, pp 227-235.

Evans R A (1985), *A Self Organising, Fault Tolerant, 2-Dimensional Array*, Proc. VLSI-85, Tokyo, Japan, ed E Hoebst, North Holland 1986, pp 239-248.

Evans R A, McCanny J V and Wood K W (1985), *Wafer Scale Integration Based on Self-Organisation*, Proc. Workshop on Wafer Scale Integra-

tion, Southampton, ed C Jesshope and W Moore, Adam Hilger, 1986, pp 101-112.

Evans R A (1986), *Wafer Scale Integration of Systolic and other Two-dimensional Processor Arrays*, proc IFIP Workshop on Wafer Scale Integration, Grenoble, March 1986.

Evans R A and McWhirter J G (1987), *A Hierarchical Testing Strategy for Self-organising Fault-tolerant Arrays*, in 'Systolic Arrays', eds Moore, McCabe and Urquhart, Adam Hilger (Bristol) UK, pp 229-238.

Evans R A (1989a), *Wafer Scale Integration* in Design and Test Techniques for VLSI and WSI Circuits, R E Massara ed., Peter Peregrinus Ltd, London, to be published 1989.

Evans R A (1989b), *Self-Organising Arrays for Wafer Scale Integration* in Design and Test Techniques for VLSI and WSI Circuits, R E Massara ed., Peter Peregrinus Ltd, London, to be published 1989.

Ferris-Prabhu A V, Smith L D, Bonges H A and Paulsen J K (1987), *Radial Yield Variations in Semiconductor Wafers*, IEEE Circuits and Devices Magazine, Vol 3, No 2, March 1987, pp 42-47.

Fitzgerald B F and Thoma E P (1980), *Circuit Implementation of Fusible Redundant Addresses in RAMs for Productivity Enhancement*, IBM Jornal or Research and Development, Vol 24, No 3, pp 291-298.

Flynn M J (1972), *Some Computer Organisations and their Effectiveness*, IEEE Trans on Computers, C-21, pp 948-960.

Franzon P (1986), *Interconnect Strategies for Fault Tolerant 2D VLSI Arrays*, proc ICCD, 1986.

Frohwerk R A (1977), *Signature Analysis - a new Digital Field Service Method*, Hewlatt Packard Journal, May 1977, pp 2-8.

Gaverick S L and Pierce E A (1983), *A Single Wafer 16-point 16 MHz FFT Processor*, Proc 1983 Custom Integrated Circuits Conference, IEEE, pp 244-248.

Grinberg J et al (1984), *3D Computing Structures for High Throughput Information Processing*, in VLSI Signal Processing, P R Cappello ed, 1984, pp 2-14.

Gupta A and Lathrop J W (1972), *Yield Analysis of Large Integrated Circuit Chips*, IEEE J. Solid-State Circuits, vol SC-7, pp 389-395, Oct 1982.

Hargrave P J (1986) *A VHPIC Node Processor*, Proc 1986 Military Microwave Conference, pp 405-410.

Hagge J K, (1988), *Ultra-reliable Packaging for Silicon-on-silicon WSI*, 38th Electronic Components Conference, May 1988.

Hedlund K S and Snyder L (1982), *Wafer Scale Integration of Configurable Highly Parallel (CHiP) Processor*, Proc 1982 Int Conference on Parallel Processing, IEEE, pp 262-264.

Hedlund K (1985), *WASP - A WAfer Scale Systolic Processor*, proc ICCD 1985, pp 665-671.

Hillis W D (1986), *The Connection Machine*, MIT Press, Cambridge, Massachusetts, ISBN 0-262-08157-1, 1986.

Hockney R W and Jesshope C R (1981), *Parallel Computers*, Adam Hilger Ltd, Bristol, UK.

Holland J H (1959), *A Universal Computer Capable of Executing and Arbitrary Number of Sub-programs Simultaneously*, Proc 16th East Joint Computer Conference, pp 108-113.

Hsia Y, Chang G C C and Erwin F D (1979), *Adaptive Wafer Scale Integration*, Proc 11th Conference on Solid State Devices, Tokyo, Jap J of Applied Physics, Vol 19, pp 193-202, Supplement 19-1.

Hu S M (1979), *Some Considerations in the Formulation of IC Yield Statistics*, Solid State Electronics, Vol 22, pp 205-211, February 1979.

Inmos Ltd (1984), *Inmos Preliminary data sheet IMS T424 Transputer*, 1984.

Katevenis M G H and Blatt M G (1985) *Switch Design for Soft-Configurable WSI Systems*, Proc 1985 VLSI Conference.

Kent F P (1983), *Yield Enhancement of Integrated Circuits by Fault Tolerant Design*, BSc Project Report, University of Southampton.

Ketchen M B (1985), *Point Defect Yield Model for Wafer Scale Integration*, IEEE Circuits and Devices Magazine, July 1985, pp 24-34.

Kobayashi A, Matsue S and Shibe H, (1968), *A Flip-flop circuit with FLT (Fault-location-technique) capability*, (in Japanese), Proc 1968 IECEO Conference, p 962.

Kuck D J (1977), *A Survey of Parallel Machine Organisation and Programming*, IEEE Trans Comput. C-17, pp 758-70.

Kung H T (1980), *Algorithms for VLSI Processor Arrays* in *Introduction to VLSI Systems* by C Mead and L Conway Addison-Wesley 1980, p 271.

Kung S Y and Gal-Ezer R J (1982), *Synchronous vs. Asynchronous Computation in VLSI Array Processors*, Proc SPIE Conference, 1982, Arlington, VA.

Landman B S and Russo R L (1971), *On Pin versus Block Relationship for Partition of Logic Graphs*, IEEE Transactions on Computers, Vol C-20, No 12, December 1971, pp 1469-1479

Lathrop J W et al (1967), *A Discretionary Wiring System as the Interface Between Desigh Automation and Semiconductor Manufacture*, Proc IEEE, Vol 55, 1967, pp 1988-1997.

Lawson T R Jr (1966), *A Prediction of the Photoresist Influence on Integrated Circuit Yield*, SCP and Solid State Technology, July 1966, pp 22-25.

MacWilliams F J and Sloane N J (1977), *Theory of Error Correcting Codes*, Elsevier, North Holland, 1977.

Mangir T E (1984), *Sources of Failures and Yield Improvement for VLSI: Part I*, Proc IEEE, Vol 72, No 2, June 1984, pp 690-708.

Manning F B (1977), *An Approach to Highly Integrated Computer-maintained Cellular Arrays*, IEEE Trans on Computers, C26, pp 536-552.

McCanny J V and McWhirter J G, (1982), *Implementation of Signal Processing Functions using 1-bit Systolic Arrays*, Electronics Letters, Vol 18, 1982, pp 241-243.

McCanny J V and McWhirter J G (1983), *Yield Enhancement of Bit-level Systolic Array Chips using Fault Tolerant Techniques*, Electronics Letters, Vol 19, No 14, July 1983, pp 525-527.

Meindl J D (1985), *Interconnection Limits on Ultra Large Scale Integration*, Proc VLSI-85, Tokyo, North Holland, pp13-19.

Meindl J D, 1987, *Chips for Advanced Computing*, Scientific American, October 1987, pp 54-62.

Moore G E (1970), *What Level of LSI is best for you?*, Electronics, Vol 43, pp 126-130, February 16 1970.

Moore W R and Day M J (1984), *Yield Enhancement of a large Systolic Array Chip*, Microelectronics and Reliability, Vol 25, No 2, 1985, pp 291-294.

Moore W R and Mahat R (1985), *Fault Tolerant Communications for Wafer Scale Integration of a Processor Array*, Microelectronics and Reliability, Vol 25, No 2, 1985, pp 291-294.

Moore W R, McCabe A P H and Bawa V (1986), *Fault Tolerance in a Large Bit-level Systolic Array*, in *Wafer Scale Integration*, C R Jesshope and W R Moore eds, Bristol UK, Adam Hilger 1986, pp 259-272.

Moore W R (1986), *A Review of Fault Tolerant Techniques for the Enhancement of Integrated circuit Yield*, IEEE Computer, Vol 745, No 5, pp 684-698.

Morison J D, Peeling N E and Thorp T L (1982), *ELLA: A Hardware Description Language*, Proc IEEE ICCC'82, September 1982, pp 604-607.

Muroga S (1982), *VLSI Systems Design*, Publ. John Wiley and Sons, pp 264-266.

Murphy B T (1964), *Cost-Size Optima of Monolithic Integrated Circuits*, Proceedings IEEE, Vol 52, December 1964, pp 1537-1545.

Murphy B T (1971), *Comments on: A New Look at Yield of Integrated Circuits*, Proc IEEE, Vol 59, July 1971 p 1128.

Nomura H (1985), *Current Status, Future Trends and Impact of VLSI*, Proc VLSI-85, Tokyo, North Holland, p 3-11.

Paz O and Lawson T R (1977), *Modification of Poisson Statistics: Modelling Defects Induced by Diffusion*, IEEE J. Solid State Circuits, Vol SC-12, October 1977, pp 540-546.

Peltzer D (1983), *Wafer Scale Integration: The Limits of VLSI?*, VLSI Design, September 1983, pp 43-47.

Peltzer D L (1983), *Wafer-Scale Integration: The Limits of VLSI?*, VLSI Design, September 1983, pp 43-47.

Perloff D S, Wahl F E, Mallory C L and Mylroie S W (1981), *Microelectronic Test Chips in Integrated Circuit Manufacturing*, Solid State Technology, Sept 1981, pp 75-80.

Petritz R I (1967), *Current Status of LSI Technology*, IEEE J of Solid State Circuits, Vol SC-2, No 4, 1967, pp 130-147.

Pitt K E G, (1987), *WSI Packaging Problems*, Proc IFIP Workshop on Wafer Scale Integration, Brunel University, 1987.

Posa J G (1981), *What to do when the Bits go out*, Electronics, July 28, pp 117-120.

Price J E (1970), *A New Look at Yield of Integrated Circuits*, Proc IEEE, Vol 58, August 1970, pp 1290-1291.

Raffel J I, Anderson A H, Chapman G H, Gaverick S L, et al (1983), *A Demonstration of very large area Integration using Laser restructuring*, Proc 1983 Int Symp. on Circuits and Systems, IEEE, pp 781-784.

Reddaway S F (1973), *DAP - A distributed array processor*, 1st annual symposium on Computer Architecture (IEEE/ACM), Florida.

Rhodes F M (1986), *Performance Characteristics of the RVLSI Technology*, Proc IFIP Workshop on Wafer Scale Integration, Grenoble, France, 1986.

Rogers T J (1982), *Redundancy in RAMs*, proc Int. Solid State Circuits Conference, February 1982, pp 228-229.

Sack K A (1964), *Evolution of the Concept of a Computer on a Slice*, Proc IEEE, Vol 52, 1964, pp 1713-1720.

Sami M and Stefanelli R (1983), *Reconfigurable Architectures for VLSI Processing Arrays*, Proc 1983 AFIPS National Computer Conference, Anaheim, California, pp 565-577.

Sami M G and Stefanelli R (1984), *Fault Tolerance of VLSI Processor Arrays - the Time-Redundant Approach*, Proc Real Time Systems Symp, IEEE, Austin.

Shaver D C (1984), *Electron-Beam Customisation, Repair and Testing of Wafer Scale Circuits*, Solid State Technology, February 1984, pp 135-139.

Shore J E (1973), *Second Thoughts on Parallel Processing*, Comput. Electrical Eng, Vol 1, pp 95-109.

Siegel H J, Siegel L J, Kemmerer F C, Mueller P T Jr, Smalley H E Jr and Smith S D (1981), *PASM: A Partitionable SIMD/MIMD System for Image Processing and Pattern Recognition*, IEEE Trans on Computers, Vol C-30, No 12, 1981, pp 934-946.

Siewiorek D P and Swarz R S (1982), *The Theory and Practice of Reliable System Design*, Digital Press, 1982, ISBN: 0-932376-13-4.

Slotnick D L, Borck W C and McReynolds R C (1962), *The SOLOMON Computer*, AFIPS Conf proc. 22, pp 97-107.

Smith R T (1981), *Using a laser beam to substitute good cells for bad*, Electronics, July 28, 1981, pp 131-134.

Stapper C H, Armstrong F M and Saji K (1983), *Integrated Circuit Yield Statistics*, Proc IEEE, Vol 71, No 4, April 1983, pp 453-470.

Stapper C H (1985), *The Effects of Wafer to Wafer Defect Density Variations on Integrated Circuit Defect and Fault Distributions*, IBM J. of Research and Development, Vol 29, January 1985, pp 87-97.

Stapper C H (1986), *On Yield, Fault Distributions and Clustering Particles*, IBM Journal of Research and Development, Vol 30, No 3, May 1986.

Stewart J H, (1977), *Future Testing of Large LSI Circuit Cards*, Semiconductor Test Symposium, October 1977, pp 6-15.

Trilhe J and Saucier G (1987), *An European program on Wafer Scale Integration*, proc VLSI-87, Vancouver, August 1987.

Turley A P and Herman D S (1974), *LSI Yield Predictions based upon Test Pattern Results: an Application to Multi-level Metal Structures*, IEEE Trans Parts, Hybrids, Packaging, Vol PHP-10, Dec 1974, pp 230-234.

Unger S H (1958), *A computer oriented towards spatial problems*, Proc Inst Radio Eng (USA), 46, pp 1744-50.

Urquhart R B and Wood D, (1984), *Systolic Matrix and Vector Multiplication Methods for Signal Processing*, Proc IEE part F, Vol 131, 1984, pp 623-631.

Wakerley J (1978), *Error Correcting Codes, Self Checking Circuits and Applications*, Elsevier, North-Holland.

Warren K D, Abdelrazik M B E, McKirdy R D and Lea R M, (1986), *A Power Distribution Strategy for WSI*, in *Wafer Scale Integration*, eds Moore and Jesshope, Adam Hilger, 1986, pp 54-61.

Warren K D and Lea R M, (1987), *Electrical Design Issues for Wafer Scale Integration*, proc IFIP Workshop on Wafer Scale Integration, Brunel University, Sept 1987.

Williams M J Y and Angell J B (1973), *Enhancing testability of Large Scale Integrated Circuits via test points and additional Logic*, IEEE Transactions on Computers 1973, pp 46-60.

Yanagawa T (1972), *Yield Degradation of Integrated Circuits due to Spot Defects*, IEEE Trans. on Electronic Devices, Vol ED-19, No 2, Feb 1972, pp 190-197.

von Neumann J (1966), *A System of 29 states with a general transition rule*, in *Theory of Self-reproducing Automata*, ed A W Burks (Urbana, Illinois: University of Illinois press) pp 305-17. (First published in 1952).

Appendix A

WINNER Performance Simulation Program

This appendix contains a listing of the program used for simulating the performance of the WINNER 3-neighbour algorithm. It is written in ALGOL-68RS and was run on a DEC-VAX 8600. Comments have been included in the text to aid in its understanding.

A.1 Program Overview

The program allows statistical data on the WINNER self-configuring algorithm to be gathered for different sizes of target array, different values of processing element yield, and for a variety of values of processing element overhead.

The user specifies a target array size, range of overhead, range of processor yield and the number of samples, N, to be evaluated and averaged for each position in the table of results. Then, for each combination of overhead and processor yield the program configures N arrays and records the number of arrays from which a functional array at least equal to the target array can be configured. Finally, these values are then plotted as a table.

In order to minimise the CPU time required, the main program has two distinct parts. The first part, represented by the procedure 'find band', searches for the band of results which separates the 0% values from the 100%

values in the table of results. Once found, a recursive procedure, 'evaluate band', is called which evaluates the band region only. This approach reduces the CPU time by about a factor of about 4 compared with that required to evaluate every point in the results table. The configuration of each array using the 3-neighbour WINNER algorithm is carried out by the procedure called 'findrows'.

A.2 Suitability for Other Algorithms

Although this particular program is only suitable for simulating the 3-neighbour WINNER algorithm, its general structure has been used to simulate other algorithms to produce the results given in chapter 6. This was done by altering the procedure 'findrows' to correspond to the required algorithm, with the number of configured rows being recorded by 'functrow'.

A.3 Program Listing

```
PROGRAM statistical configuration

CO  Simulation of the WINNER algorithm with three-neighbour
    connectivity.

VARIABLES

    physrow    - row-number of physical rows on the array.
    functrow   - row-number of functional rows on the array.
    rows       - target for number of rows to be configured.
    cols       - target for number of columns to be configured.
    randstart  - parameter used by 'nextrandom'.
    chipyield  - percentage yield of cells in array.
    cell       - array of elements to be configured.
    minovhd    - minimum value of the overhead range (in rows).
    maxovhd    - maximum value of the overhead range (in rows).
    minyield   - minimum value of the yield range (in %).
    maxyield   - maximum value of the yield range (in %).
    yieldstep  - incremental yield between minyield and maxyield.
    iterations- number of samples for each result in the table.
CO

BEGIN

INT physrow, functrow, rows, cols, randstart;
INT minovhd, maxovhd, minyield, maxyield, yieldstep, iterations;
INT ovhdrange, yieldrange, ovhdelem, yieldelem;
CHAR file;

MODE CELL = STRUCT(BOOL failure, avail, INT rowno);


{ ********************* Read in data ********************* }

print(("Enter target array in rows and columns",newline));
read((rows, cols));
print(("Enter min and max overhead in rows",newline));
read((minovhd, maxovhd));
print(("Enter yield range and step (%)",newline));
read((minyield, maxyield, yieldstep));
print(("Enter number of iterations required",newline));
read(iterations);
print(("Output to file? <y or n>",newline));
read(file);
```

```
{ ************ Set up output file and channel ************ }

FILE arrayout;
REF FILE current = IF file = "y"
                   THEN INT est;
{ outputfile is logical name }
                      est:=establish(arrayout, "outputfile",
                                standoutchannel,1,1000,150);
                      IF est/=0
                      THEN fault(est,"Fault in establish")
                      FI;
                      arrayout
                   ELSE standout
                   FI;


{ ************* Set up data dependent arrays ************** }

ovhdrange  := maxovhd-minovhd+1;
yieldrange := (maxyield-minyield)%yieldstep+1;

{ Set up array to hold results as they are generated }
[ovhdrange,yieldrange]REAL bin;

{ Set up array to record which points in result table have }
{ been evaluated. }
[ovhdrange,yieldrange]BOOL been;

clear(bin);
clear(been);


{ ********* Print out initial data to output file ********* }

putf(current, ($ "Target array size = " 2zd," rows by"2zd"
                 columns."21 $, rows,cols));
putf(current, ($ "Overhead Range = "2zd," rows to "2zd"
                 rows."21 $, minovhd,maxovhd));
putf(current, ($ "Yield Range = "2zd," to "2zd," %, Step = "2zd"
                 %"21 $, minyield,maxyield,yieldstep));
putf(current, ($ "Number of Iterations = 3zd 21 $, iterations));


{ ********** Start of procedure declarations *********** }

PROC setfaults = (REF[,]CELL cell, INT ovhd, chipyield)VOID:
{ Sets up random fault distribution for array to be configured }
```

```
   BEGIN
      INT goodrow, goodcol, goodcells, goodposition;
      INT actualrows:=rows+ovhd;
      FORALL cc IN cell
         DO FORALL c IN cc DO failureOFc:=TRUE OD OD;
      goodcells := ENTIER(actualrows*cols*chipyield/100);
      WHILE goodcells/=0
        DO goodposition:=
              ENTIER(nextrandom(randstart)*actualrows*cols+1);
           goodcol:=(goodpositionMODcols);
           IF goodcol=0 THEN goodcol:=cols FI;
           goodrow:=ENTIER((goodposition-goodcol)/cols)+1;
           IF NOT failureOFcell[goodrow,goodcol]
             THEN SKIP
             ELSE failureOFcell[goodrow,goodcol]:=FALSE;
                  goodcells MINUSAB 1
           FI
        OD
   END;

PROC setedges = (REF[,]CELL cell, INT ovhd)VOID:
{ Sets up edge conditions in array to be configured }
   BEGIN
      INT actualrows:=rows+ovhd;
      FOR i FROM 0 TO actualrows+1
         DO
           FOR j FROM 0 TO cols+1
              DO availOFcell[i,j]:=
                    IF j=0 OREL i=0 OREL i=actualrows+1
                       THEN FALSE
                       ELSE TRUE
                    FI
              OD
         OD
   END;

PROC updateavail = (REF [,]CELL cell, INT ovhd)VOID:
{ Updates availability signals after each functional }
{ row is established. }
   BEGIN
      INT actualrows:=rows+ovhd;
      FOR j FROM cols BY -1 TO 1
         DO FOR i TO actualrows
               DO availOFcell[i,j]:=
                     availOFcell[i,j] ANDTH
                     (availOFcell[i+1,j+1]  OREL
                      availOFcell[i  ,j+1]  OREL
                      availOFcell[i-1,j+1]) ANDTH NOT
```

```
                failureOFcell[i,j]
            OD
        OD
    END;

PROC findrows = (REF[,]CELL cell, INT ovhd)VOID:
{ Finds as many functional rows as possible }
    BEGIN
        INT actualrows:=rows+ovhd;
        REF CELL ptr := NIL;
        { clear(rownoOFcell); }
        FORALL cc IN cell
            DO FORALL c IN cc DO rownoOFc:=0 OD OD;
        functrow:=0;
        FOR i TO actualrows
            DO updateavail(cell,ovhd);
                IF availOFcell[i,1]
                    THEN physrow:=i;
                        functrow PLUSAB 1;
                        rownoOFcell[i,1]:=functrow;
                        availOFcell[i,1]:=FALSE;
                        FOR rhcell FROM 2 TO cols
                            DO IF ptr := cell[physrow-1,rhcell];
                                availOFptr
                               THEN physrow MINUSAB 1
                               ELIF ptr := cell[physrow,rhcell];
                                    availOFptr
                               THEN SKIP
                               ELSE ptr := cell[physrow+1,rhcell];
                                    physrow PLUSAB 1
                              FI;
                              availOFptr := FALSE;
                              rownoOFptr := functrow
                            OD
                FI
            OD
    END;

PROC printarray = (REF[,]CELL cell, INT ovhd, chipyield)VOID:
{ Prints configured array }
    BEGIN
        INT actualrows:=rows+ovhd;
        putf(current, ($"Chip Yield = " 3zd.d " Percent"
                      2l$,chipyield));
        putf(current, ($"Number of Functional Rows found =
                      "3zd 2l$,functrow));
        putf(current,  $ n(cols)(aq)l $);
        FOR i TO actualrows
```

```
        DO FOR j TO cols
         DO putf(current,
                     IF failureOFcell[i,j] THEN "x"
                     ELIF rownoOFcell[i,j]=0 THEN "."
                     ELSE whole(rownoOFcell[i,j] MOD 10,0)
                     FI)
         OD
        OD
   END;

PROC configure array =
         (REF [,]CELL cell, INT ovhd, chipyield)VOID:
{ Sets up and configures a single complete array }
   BEGIN
      setedges(cell,ovhd);
      setfaults(cell,ovhd,chipyield);
      findrows(cell,ovhd)
   END;

PROC evaluate point = (INT ovhdelem, yieldelem)VOID:
{ Evaluates one point in the table of results }
   BEGIN
      INT ovhd:=ovhdelem+minovhd-1;
      [0:rows+ovhd+1, 0:cols+1]CELL cell;
      INT chipyield:=minyield+yieldstep*(yieldelem-1);
      TO iterations
         DO configure array(cell,ovhd,chipyield);
            IF functrow>=rows
               THEN bin[ovhdelem,yieldelem] PLUSAB 1
            FI
         OD;
         been[ovhdelem,yieldelem]:=TRUE
   END;

PROC find band = VOID:
{ Searches for band of non-extreme values in table of results }
   BEGIN
      WHILE evaluate point(ovhdelem,yieldelem);
            IF   bin[ovhdelem,yieldelem]=0
                 THEN ovhdelem PLUSAB 1;
                      TRUE
            ELIF bin[ovhdelem,yieldelem]=iterations
                 THEN yieldelem MINUSAB 1;
                 TRUE
            ELSE FALSE
            FI
      DO SKIP OD
   END;
```

```
PROC evaluate band = (INT ovhdelem, yieldelem)VOID:
{ Evaluates all points within non-extreme band }
   BEGIN
      IF yieldelem>1
         THEN IF NOT been[ovhdelem,yieldelem-1] ANDTH
                     bin[ovhdelem,yieldelem]/=0
                 THEN evaluate point(ovhdelem, yieldelem-1);
                      evaluate band(ovhdelem,yieldelem-1)
              FI
      FI;

      IF ovhdelem>1
         THEN IF NOT been[ovhdelem-1,yieldelem] ANDTH
                     bin[ovhdelem,yieldelem]/=0
                 THEN evaluate point(ovhdelem-1, yieldelem);
                      evaluate band(ovhdelem-1,yieldelem)
              FI
      FI;

      IF yieldelem<yieldrange
         THEN IF NOT been[ovhdelem,yieldelem+1] ANDTH
                     bin[ovhdelem,yieldelem]/=iterations
                 THEN evaluate point(ovhdelem,yieldelem+1);
                      evaluate band(ovhdelem,yieldelem+1)
              FI
      FI;

      IF ovhdelem<ovhdrange
         THEN IF NOT been[ovhdelem+1,yieldelem] ANDTH
                     bin[ovhdelem,yieldelem]/=iterations
                 THEN evaluate point(ovhdelem+1, yieldelem);
                      evaluate band(ovhdelem+1,yieldelem)
              FI
      FI
   END;

PROC printresults = VOID:
{ Prints table of results }
   BEGIN
      put(current, ("SIMULATION OF WINNER WITH CONNECTIONS
                     TO 3 NEIGHBOURS",
                     newline, newline));
      put(current, ("PERCENTAGE ARRAY YIELD AS A", newline));
      put(current, ("FUNCTION OF OVERHEAD AND PROCESSOR YIELD",
                     newline,newline));

      put(current, ("    OVERHEAD (ROWS)", newline,newline));
```

```
      putf(current, $2zdx "|", n(yieldrange)(x2z.z) 1$);
      FOR ovhd FROM maxovhd BY -1 TO minovhd
         DO putf(current, ovhd);
            FOR yieldelem TO yieldrange
               DO putf(current,
                       IF ENTIER bin[ovhd-minovhd+1,yieldelem]
                          MOD 100 = 0
                          THEN 0
                          ELSE bin[ovhd-minovhd+1,yieldelem]
                       FI)
               OD
         OD;
      putf(current, $4x n(5*yieldrange+1)"-" l 6x
                      n(yieldrange)(2zd2x) l 4x
                      n(yieldrange*5%2-9)x
                      "PROCESSOR YIELD (%)"$);
      FOR yield FROM minyield BY yieldstep TO maxyield
         DO putf(current, yield) OD
   END;

{ ****************** Main Program ****************** }


{ Set starting point for ovhdelem and yieldrange }
ovhdelem  := 1;
yieldelem := yieldrange;

find band;
evaluate band(ovhdelem, yieldelem);

{ Convert results to percentages }
FOR m TO ovhdrange
   DO FOR n TO yieldrange
         DO bin[m,n]:=bin[m,n]/iterations*100 OD
   OD;

printresults

END

FINISH
```

Appendix B

ELLA Simulation Program

This file contains the ELLA description of the hardware required to implement the *WINNER* algorithm. The array of *WINNER* cells has been described in parameterised form so that arrays of differing sizes can be simulated relatively easily. A simple processing element is used so that program complexity is kept to a minimum. The function of each processor in both the horizontal and vertical directions is to add one to an incoming digit and pass on the results to neighbouring cells.

B.1 Program Listing

```
INT m = 6,
    n = 6.
# m and n determine the size of the array #
# as defined by the macro 'array' #

TYPE bool = NEW (t|f|x),
     data = NEW da/(0..200),
     limint = NEW i/(1..n).
COM
The gates which carry out the boolean functions AND, NOT
and OR are defined below. Gates MUX_AND and MUX_OR are based
on their boolean equivelents, but act as multiplexers such
that all bar one of the inputs are controls and determine
the output (which is an ELLA integer type). TWO_IP_MUX
defines a multiplexer with two inputs and a control line,
and PROCESSOR defines a mathematical function (in this case,
to add one to an ELLA integer).
MOC
```

```
#----------AND----------#
FN AND = (bool: input1 input2) -> bool:
CASE (input1, input2)
OF (t, t): t,
   (f, bool)|(bool, f): f
ELSE x
ESAC.

#----------NOT----------#
FN NOT = (bool: input) -> bool:
CASE input
OF f: t,
   t: f
ELSE x
ESAC.

#----------THREE_IP_AND----------#
FN THREE_IP_AND = ([3]bool: ip) -> bool:
CASE ip OF
      (t, t, t): t,
      (f, bool, bool) | (bool, f, bool) | (bool, bool, f): f
ELSE x
ESAC.

#----------THREE_IP_OR----------#
FN THREE_IP_OR = ([3]bool: ip) -> bool:
CASE ip OF
      (f, f, f): f,
      (t, bool, bool) | (bool, t, bool) | (bool, bool, t): t
ELSE x
ESAC.

#----------FOUR_IP_AND----------#
FN FOUR_IP_AND = ([4]bool: ip) -> bool:
CASE ip
  OF (f, bool, bool, bool) | (bool, f, bool, bool)|
      (bool, bool, f, bool) | (bool, bool, bool, f): f,
      (t, t, t, t): t
ELSE x
ESAC.



#----------MUX_AND----------#
FN MUX_AND = (bool: ip1, data: ip2) -> data:
CASE ip1
```

```
  OF t: ip2,
     f: da/0
ELSE ?data
ESAC.

#----------MUX_OR----------#
FN MUX_OR = ([3]data: ip) -> data:
CASE (ip[1]>da/0, ip[2]>da/0, ip[3]>da/0)
  OF (t, f, f): ip[1],
     (f, t, f): ip[2],
     (f, f, t): ip[3]
ELSE ?data
ESAC.

#----------TWO_IP_MUX----------#
FN TWO_IP_MUX = (data: ip1 ip2, bool: ctrl) -> data:
CASE ctrl
OF t: ip1,
   f: ip2
ESAC.

#----------PROCESSOR----------#
# The boolean input 'passfail' determines whether #
# or not the cell is working #

FN PROCESSOR = (data: ip1 ip2, bool: ip3)
            -> (data, data, bool):
BEGIN
     LET out1 = ip1 + da/1.
     LET out2 = ip2 + da/1.
     LET passfail = ip3.
     OUTPUT (out1, out2, passfail)
END.

#----------CELL----------#

COM
This function connects the processor to the control
logic (which comprises those gates defined),such that
each cell is able to communicate with its neighbours.
MOC

FN CELL = (data: ipn ipnw ipw ipsw,
           bool: reqnw reqw reqsw
                 availse availe availne pf) ->
          ([4]data,
          # ops  opse  ope   opne #
           [6]bool
```

```
        # availnw availw availsw reqse reqe reqne #):
BEGIN
    MAKE PROCESSOR: processor,
         TWO_IP_MUX: two_ip_mux,
         [6]AND: and,
         [3]MUX_AND: mux_and,
         THREE_IP_AND: three_ip_and,
         FOUR_IP_AND: four_ip_and,
         [2]THREE_IP_OR: three_ip_or,
         MUX_OR: mux_or,
         [4]NOT: not.







    JOIN (ipn, mux_or, pf) -> processor,
         ( processor[1], ipn,  and[5]) -> two_ip_mux,
         (mux_and[1], mux_and[2], mux_and[3]) -> mux_or,
         reqnw -> not[1],
         (not[1], and[3]) -> and[1],
         reqw -> not[2],
         (not[2], and[1]) -> and[2],
         (reqnw, ipnw) -> mux_and[1],
         (reqw, ipw) -> mux_and[2],
         (reqsw, ipsw) -> mux_and[3],
         (reqnw, reqw, reqsw) -> three_ip_or[1],
         (processor[3], three_ip_or[1]) -> and[4],
         (processor[3], three_ip_or[2]) -> and[3],
         (and[3], three_ip_or[1]) -> and[5],
         (availne, availe, availse) -> three_ip_or[2],
         availne -> not[3],
         (not[3], and[4], availe) -> three_ip_and,
         availe -> not[4],
         (not[4], not[3], and[4], availse) -> four_ip_and,
         (and[4], availne) -> and[6].
    OUTPUT ((two_ip_mux, processor[2], processor[2],
             processor[2]), (and[3], and[1], and[2],
             four_ip_and, three_ip_and, and[6]))
END.

#----------MACRO: ARRAY----------#

COM
The macro ARRAY defines an array of cells with m
```

```
columns and n rows with inputs for those cells on the
edges of the array. FN ARRAY_OF_CELLS creates m x n
cells which are then connected together systematically,
each direction being considered separately.
MOC

MAC ARRAY{INT m n} = ([m]data: ipn, [m+n-1]data: ipnw ipsw,
                       [n]data: ipw,
                       [m+n-1]bool: reqnw reqsw availse availne,
                       [n]bool: reqw availe, [m][n]bool:pf) ->
                      ([m][n]data, [m][n]data):
                     # ops       ops #

 BEGIN
   FN ARRAY_OF_CELLS = ([m][n][4]data:ip1, [m][n][7]bool:ip2)
                      -> [m][n]([4]data, [6]bool):
     [INT i=1..m][INT j=1..n]
                      CELL (ip1[i][j][1], ip1[i][j][2],
                            ip1[i][j][3], ip1[i][j][4],
                            ip2[i][j][1], ip2[i][j][2],
                            ip2[i][j][3], ip2[i][j][4],
                            ip2[i][j][5], ip2[i][j][6],
                            ip2[i][j][7]).

COM
In connecting the cells together, each cell is
considered separately for each direction. If the
cell is on the edge of the array, input connections
are made, otherwise the cell is connected to adjacent ones.
MOC

     MAKE ARRAY_OF_CELLS: array.
     JOIN ([INT i=1..m][INT j=1..n]
          (IF j=1 THEN ipn[i]
                 ELSE array[i][j-1][1][1] FI,    # ipn #
           IF j=1 THEN ipnw[i]
                 ELSE IF i=1 THEN ipnw[j]
                             ELSE array[i-1][j-1][1][2]
                      FI
           FI,                                    # ipnw #


           IF i=1 THEN ipw[j]
                 ELSE array[i-1][j][1][3] FI,    # ipw #
           IF j=n THEN ipsw[i]
                 ELSE IF i=1 THEN ipsw[j]
                             ELSE array[i-1][j+1][1][4]
```

```
                    FI
          FI                                            # ipew #
         ),
         [INT i=1..m][INT j=1..n]
         (IF j=1 THEN reqnw[i]
                 ELSE IF i=1 THEN reqnw[j]
                             ELSE array[i-1][j-1][2][4]
                      FI
          FI,                                           # reqnw #
          IF i=1 THEN reqw[j]
                 ELSE array[i-1][j][2][5] FI,           # reqw #
          IF j=n THEN reqsw[j]
                 ELSE IF i=1 THEN reqsw[j]
                             ELSE array[i-1][j+1][2][6]
                      FI
          FI,                                           # reqsw #
          IF j=n THEN availse[i]
                 ELSE IF i=m THEN availse[j]
                             ELSE array[i+1][j+1][2][1]
                      FI
          FI,                                           # availse #
          IF i=m THEN availe[j]
                 ELSE array[i+1][j][2][2] FI,           # availe #
          IF j=1 THEN availne[i]
                 ELSE IF i=m THEN availne[j]
                             ELSE array[i+1][j-1][2][3]
                      FI
          FI,                                           # availne #
          pf[j][i])) -> array.
 LET colop = [INT i=1..m][INT j=1..n]array[i][j][1][1].
     rowop = [INT j=1..n][INT i=1..m]array[i][j][1][3].

COM
The output format is to output the signals from
each cell in the easterly and southerly directions.
'colop' stores the southerly outputs in columns, and
'rowop' stores the easterly outputs in rows.
MOC


OUTPUT (colop, rowop)

END.

#----------COUNTER----------#

COM
This function basically defines a ring counter which
```

```
starts at 1 and counts one more for every time cycle.
When 12 is reached, the counter loops back to 1 on
the following time cycle.
MOC

FN COUNTER = (bool) -> data:
BEGIN SEQ
         STATE VAR count INIT da/O;
         FN INC = (data: ip) -> data:
            ARITH IF ip=(2*n+1) THEN 1
                          ELSE (ip + 1) FI;
         LET out = count;
         count := INC(out);
         OUTPUT (out)
END.

#----------LIMIT----------#
#This function ensures indexing in 'ROWMUX' #
# does not become 0 #

 FN LIMIT = (data: ip) -> limint: ARITH ip.

#---------- < ----------#

FN < = (data: a b) -> bool: ARITH IF a<b THEN 1
                                          ELSE 2 FI.
#---------- '-' ----------#

FN - = (data: a b) -> data: ARITH a-b.

#---------- = ----------#

FN = = (data: a b) -> bool: ARITH IF a=b THEN 1
                                          ELSE 2 FI.

#----------MAC: ROWMUX----------#

COM
This macro multiplexes the output rows of data
from 'ARRAY'. Successive rows are output each
time unit by way of the increasing count (select).
MOC

MAC ROWMUX{INT m n) = ([m][n]data: ip1, [m][n]data: ip2)
                    -> [m]data:
BEGIN
     LET select = COUNTER(t).
     OUTPUT
```

```
        CASE select=da/1
        OF t: ([INT j=1..m]da/111)
        ELSE CASE select<da/8
              OF t: ([INT j=1..m]ip2[[LIMIT(select-da/1)]][j]),
                 f: ([INT j=1..m]ip1[j][[LIMIT(select-da/7)]])
              ESAC
        ESAC
END.


#----------CONFIGURE_ARRAY----------#

COM
This function configures the array of cells. To change
the dimensions of the array, different values are
given to m and n (which are defined as INTegers at the
start of the program).
MOC

FN CONFIGARRAY = ([m]data:ip1, [n]data:ip2, [m][n]bool:ip3)
                 -> ([m]data):
BEGIN
       LET diagonal=(ip1,[m+n-1]da/1,[m+n-1]da/1,ip2,[m+n-1]f,
                      [m+n-1]f,[m+n-1]f,[m+n-1]f,[n]t,[n]t,ip3).
       MAKE ARRAY{m, n}: array,
            ROWMUX{m, n}: rowmux.
       JOIN diagonal -> array,
            array -> rowmux.
       OUTPUT rowmux
END.
```

Appendix C

WINNER Demonstrator Test Program

C.1 Program Overview

This program, written in BASIC, and run on a BBC computer, controls the *WINNER* demonstrator array. All inputs to the array are generated by the program, and all outputs from the array are monitored. The main purpose of the program is to provide the test patterns for the scan path test of the control circuitry. In addition, the program allows selection between several modes of operation, such as normal configuration, test, and control circuit fault masking. The program itself is controlled by a menu, from which the user can select the required mode of operation.

The program make full use of procedures which makes it relatively easy to understand. However, comments have been incorporated where appropriate.

C.2 Program Listing

```
 5 REM *** MAIN PROGRAM ***
10 CLS
20 PROCSETUP
30 PROCREFGEN
40 PROCMENU
50 IF KEY=1 THEN PROCFILL : PROCCONFIG
60 IF KEY=2 THEN 30
70 IF KEY=3 THEN PROCFINDFAULTS : PROCMASKINIT :
```

```
    PROCCHECKTEST : PROCMASKGEN : PROCLOADMASK : PROCCONFIG
 75
 76 REM *** PROCEDURE DECLARATIONS ***
 77
 80 DEF PROCSETUP
 85 REM *** SETS UP THE NECESSARY OUTPUT PORTS AND ARRAYS ***
 90 A=0:B=64:C=128:D=192
100 ?&FE62=195 :REM DDRB SET
110 ?&FE66=&FF :REM DDRA OUT
120 ?&FE6C=&0A :REM PULSE OUT
130 ?&FE61=0   :REM CLEAR LSB
140 ?&FE61=64  :REM CLEAR
150 ?&FE61=128 :REM CLEAR
160 ?&FE61=192 :REM CLEAR MSB
170 DIM ARRAY(4,48)
180 DIM ARRAY2(4,48)
190 DIM RESULTS(4,48)
200 DIM MASK(48)
210 DIM FMASK(48)
215 VCLK=?&FE60
216 ?&FE60=(VCLK OR 3)
220 ENDPROC
230
240 DEF PROCREFGEN
245 REM *** APPLIES TEST TO FAULT-FREE ARRAY AND ***
246 REM *** STORES OUTPUTS FOR USE AS REFERENCES ***
247 REM *** FOR COMPARISON WITH FAULTY OUTPUTS ***
250 FOR Q=1 TO 4
260 FOR Q1=1 TO 48
270 ARRAY(Q,Q1)=0
280 RESULTS(Q,Q1)=0
290 NEXTQ1
300 NEXT Q
310 CLS
320 PRINT "GENERATING REFERENCE ARRAY"
330 PRINT:PRINT
340 PROCAPPLYTEST
350 FOR I=1 TO 4
360 FOR J=1 TO 48
370 ARRAY(I,J)=RESULTS(I,J)
380 NEXT J
390 NEXT I
400 ENDPROC
410
420 DEF PROCFINDFAULTS
425 REM *** APPLIES TEST PATTERN TO ARRAY TO ***
426 REM *** FIND CTRL CCT FAULTS ***
430 FOR Q=1 TO 4
```

```
440 FOR Q1=1 TO 48
450 ARRAY2(Q,Q1)=0
460 RESULTS(Q,Q1)=0
470 NEXTQ1
480 NEXT Q
490 CLS
500 PRINT "GENERATING FAULT ARRAY"
510 PRINT:PRINT
520 PROCAPPLYTEST
530 FOR I=1 TO 4
540 FOR J=1 TO 48
550 ARRAY2(I,J)=RESULTS(I,J)
560 NEXT J
570 NEXT I
580 ENDPROC
590
600 DEF PROCFILL
601 REM *** FILL SCAN PATHS WITH 1'S TO ALLOW ***
602 rem *** NORMAL CONFIGURATION ***
605 PROCAOFF
606 PROCBOFF
620 PROCSINSON
630 FOR F=1 TO 100
640 PROCCLK
650 NEXT F
660 PROCAON
665 PROCBON
670 ENDPROC
680
690 DEF PROCAPPLYTEST
700 REM *** APPLY TEST PATTERN TO THE ARRAY ***
710 PRINT "        PLEASE WAIT"
720 PRINT :PRINT
730 DATA 0,192,36,3
740 PROCAOFF
750 PROCBOFF
760 RESTORE
770 FOR PATNUM=0 TO 3
780 READ CNT
790 PRINT "APPLYING TEST PATTERN NUMBER ";PATNUM+1
800 SLICENUM=1
810 FOR ROW=1 TO 6
820 BIT = 1
830 FOR BITPOS=1 TO 8
840 IF (CNT AND BIT)=0 THEN PROCSINSOFF ELSE PROCSINSON
850 PROCREAD
860 PROCCLK
870 BIT=BIT*2
```

```
 880 SLICENUM=SLICENUM+1
 890 NEXT BITPOS
 900 NEXT ROW
 910 PROCAON
 915 PROCDELAY
 920 PROCSLOCLK
 930 PROCSLOCLK
 940 PROCCLK
 945 PROCDELAY
 950 PROCAOFF
 960 NEXT PATNUM
 970 PROCLAST48
 980 ENDPROC
 990
1000 DEF PROCMASKINIT
1005 REM *** INITIALISE FAULT MASKING ARRAY TO 1'S ***
1010 FOR POSN=1 TO 48
1020 MASK(POSN)=255
1030 FMASK(POSN)=255
1040 NEXT POSN
1050 ENDPROC
1060
1070 DEF PROCCHECKTEST
1075 REM *** COMPARE RESULTS OF TEST WITH REFERENCE ***
1080 FOR COL=1 TO 4
1090 FOR ROW=1 TO 48
1100 P=ARRAY(COL,ROW) EOR ARRAY2(COL,ROW)
1110 P=NOT P
1120 P=(P AND (MASK(49-ROW)))
1130 MASK(49-ROW)=P
1140 NEXT ROW
1150 NEXT COL
1160 ENDPROC
1170
1180 DEF PROCSINSON
1185 REM *** SET GROUP OF SUM INPUTS TO LOGIC 1 ***
1190 PROCS1ON
1200 PROCS2ON
1210 PROCS3ON
1220 PROCS4ON
1230 PROCS5ON
1240 ENDPROC
1250
1260 DEF PROCSINSOFF
1265 REM *** SET GROUP OF SUM INPUTS TO LOGIC 0 ***
1270 PROCS1OFF
1280 PROCS2OFF
1290 PROCS3OFF
```

```
1300 PROCS4OFF
1310 PROCS5OFF
1320 ENDPROC
1330
1340 DEF PROCREAD
1341 REM *** READ INFO FROM THE ARRAY INTO COMPUTER ***
1345 VREAD=?&FE60
1350 ?&FE60=(VREAD OR 128)
1355 VREAD=?&FE60
1360 ?&FE60=(VREAD AND 191)
1370 AB=(?&FE60 AND 60)/4
1380 VREAD=?&FE60
1385 ?&FE60=(VREAD OR 192)
1390 CD=(?&FE60 AND 32)/2
1400 ANS = AB OR CD
1410 RESULTS(PATNUM,SLICENUM)=ANS
1420 ENDPROC
1430
1440 DEF PROCLAST48
1445 REM *** CLOCK OUT LAST 48 BITS OF TEST RESPONSE ***
1450 PATNUM=4
1460 FOR SLICENUM=1 TO 48
1470 PROCREAD
1480 PROCCLK
1490 NEXT SLICENUM
1500 ENDPROC
1510
1520 DEF PROCMASKGEN
1525 REM *** GENERATE ARRAY TO MASK CTRL CCT FAULTS ***
1530 FOR COL=1 TO 4
1540 LINDEX=2^(5-COL)
1550 RINDEX=2^(4-COL)
1560 INDOR=LINDEX OR RINDEX
1570 FOR ROW=1 TO 6
1580 OSET=((ROW-1)*8)+1
1590 IF ROW<>1 THEN AVAILNW=(MASK(OSET-1))AND LINDEX
     ELSE AVAILNW=LINDEX
1600 AVAILW=(MASK(OSET+5))AND LINDEX
1610 IF ROW<>6 THEN AVAILSW=(MASK(OSET+9))AND LINDEX
     ELSE AVAILSW=LINDEX
1620 REQNE=(MASK(OSET))AND RINDEX
1630 REQE=(MASK(OSET+2))AND RINDEX
1640 REQSE=(MASK(OSET+6))AND RINDEX
1650 LAND=AVAILNW AND AVAILW AND AVAILSW
1660 RAND=REQNE AND REQE AND REQSE
1670 MASKVAL=LAND OR RAND
1680 IF MASKVAL<> INDOR THEN PROCRMASK
1690 NEXT ROW
```

```
1700 NEXT COL
1710 ENDPROC
1720
1730 DEF PROCRMASK
1735 REM *** USED IN PROCMASKGEN.  ASSIGNS MASK BIT ***
1736 REM *** TO EACH OUTPUT OF A FAULTY CELL ***
1740 PRINT"FAULT AT "
1750 PRINT"ROW ";ROW
1760 PRINT"COL ";COL
1770 PRINT"---------------"
1780 IF ROW<>1
     THEN FMASK(OSET-1)=(MASK(OSET-1))AND(NOT LINDEX)
1790 FMASK(OSET+5)=(MASK(OSET+5))AND (NOT LINDEX)
1800 IF ROW <>6
     THEN FMASK(OSET+9)=(MASK(OSET+9))AND(NOT LINDEX)
1810 FMASK(OSET)=(MASK(OSET))AND(NOT RINDEX)
1820 FMASK(OSET+2)=(MASK(OSET+2))AND(NOT RINDEX)
1830 FMASK(OSET+6)=(MASK(OSET+6))AND(NOT RINDEX)
1840 FMASK(OSET+3)=(FMASK(OSET+3))AND (NOT RINDEX)
1850 FMASK(OSET+4)=(FMASK(OSET+4))AND (NOT RINDEX)
1860 ENDPROC
1870
1880 DEF PROCLOADMASK
1885 REM *** LOAD MASK INTO ARRAY ***
1890 PROCAOFF
1900 PROCBOFF
1910 FOR I=1 TO 48
1920 P=FMASK(49-I)
1930 IF (P AND 1)=1 THEN PROCS4ON ELSE PROCS4OFF
1940 IF (P AND 2)=2 THEN PROCS3ON ELSE PROCS3OFF
1950 IF (P AND 4)=4 THEN PROCS2ON ELSE PROCS2OFF
1960 IF (P AND 8)=8 THEN PROCS1ON ELSE PROCS1OFF
1970 IF(P AND 16)=16 THEN PROCS5ON ELSE PROCS5OFF
1980 PROCCLK
1990 NEXT I
2000 PROCAON
2010 PROCBON
2020 ENDPROC
2030
2040 DEF PROCCONFIG
2045 REM *** MENU FOR CONTROLLING CONFIGURATION MODES ***
2050 CLS
2060 PRINT:PRINT:PRINT:PRINT
2070 PRINT"   1. CONTINUOUS CONFIGURE"
2080 PRINT
2090 PRINT"   2. ONE CONFIGURE CLOCK PULSE"
2100 PRINT
2110 PRINT"   3.RETURN TO MENU "
```

```
2120 PRINT:PRINT
2130  PRINT"   PLEASE SELECT OPTION ( 1,2 OR 3 )"
2140 CONKEY=GET : CONKEY=CONKEY-48
2150 IF CONKEY<1 OR CONKEY>3 THEN 2140
2160 IF CONKEY=3 THEN 40
2170 IF CONKEY=2 THEN PROCSLOCLK : GOTO 2140
2180 IF CONKEY=1 THEN PROCCONTCONF : GOTO 2050
2190
2200 DEF PROCCONTCONF
2205 REM *** ALLOWS ARRAY TO CONFIGURE CONTINUOUSLY ***
2210 CLS
2220 PRINT:PRINT:PRINT:PRINT
2230 PRINT  "CONTINUOUSLY CONFIGURING"
2240 PRINT
2250 PRINT  " PRESS ANY KEY TO STOP"
2260 IF INKEY(1)=-1 THEN PROCSLOCLK : GOTO 2260
2270 ENDPROC
2280
2290 DEF PROCMENU
2295 REM *** GENERATES TOP LEVEL MENU ***
2300 CLS
2310 PRINT:PRINT:PRINT:PRINT
2320 PRINT"-----------------MENU-----------------"
2330 PRINT
2340 PRINT"    1. W.I.N.N.E.R. ALGORITHM DEMO"
2350 PRINT
2360 PRINT"    2. SCAN REFERENCES"
2370 PRINT
2380 PRINT"    3. SCAN & MASK OUT FAULTS"
2390 PRINT:PRINT:PRINT
2400 PRINT"PLEASE SELECT OPTION ( 1,2 OR 3 )"
2410 KEY=GET:KEY=KEY-48
2420 IF KEY<1 OR KEY>3 THEN 2410
2430 ENDPROC
2435
2436 REM *** PROCEDURES TO SET ARRAY INPUT SIGNALS ***
2437
2440 DEF PROCAON
2450 A=A OR 1
2460 GOTO 3780
2470 DEF PROCAOFF
2480 A=A AND 62
2490 GOTO 3780
2500 DEF PROCBON
2510 A=A OR 2
2520 GOTO 3780
2530 DEF PROCBOFF
2540 A=A AND 61
```

```
2550 GOTO 3780
2560 DEF PROCDATN1ON
2570 A=A OR 4
2580 GOTO 3780
2590 DEF PROCDATN1OFF
2600 A=A AND 59
2610 GOTO 3780
2620 DEF PROCDATN2ON
2630 A=A OR 8
2640 GOTO 3780
2650 DEF PROCDATN2OFF
2660 A=A AND 55
2670 GOTO 3780
2680 DEF PROCDATN3ON
2690 A=A OR 16
2700 GOTO 3780
2710 DEF PROCDATN3OFF
2720 A=A AND 47
2730 GOTO 3780
2740 DEF PROCDATN4ON
2750 A=A OR 32
2760 GOTO 3780
2770 DEF PROCDATN4OFF
2780 A=A AND 31
2790 GOTO 3780
2800 DEF PROCTESTON
2810 B=B OR 65
2820 GOTO 3800
2830 DEF PROCTESTOFF
2840 B=B AND 126
2850 GOTO 3800
2860 DEF PROCS1ON
2870 B=B OR 66
2880 GOTO 3800
2890 DEF PROCS1OFF
2900 B=B AND 125
2910 GOTO 3800
2920 DEF PROCS2ON
2930 B=B OR 68
2940 GOTO 3800
2950 DEF PROCS2OFF
2960 B=B AND 123
2970 GOTO 3800
2980 DEF PROCS3ON
2990 B=B OR 72
3000 GOTO 3800
3010 DEF PROCS3OFF
3020 B=B AND 119
```

```
3030 GOTO 3800
3040 DEF PROCS4ON
3050 B=B OR 80
3060 GOTO 3800
3070 DEF PROCS4OFF
3080 B=B AND 111
3090 GOTO 3800
3100 DEF PROCS5ON
3110 B=B OR 96
3120 GOTO 3800
3130 DEF PROCS5OFF
3140 B=B AND 95
3150 GOTO 3800
3160 DEF PROCSUM1ON
3170 C=C OR 129
3180 GOTO 3820
3190 DEF PROCSUM1OFF
3200 C=C AND 190
3210 GOTO 3820
3220 DEF PROCSUM2ON
3230 C=C OR 130
3240 GOTO 3820
3250 DEF PROCSUM2OFF
3260 C=C AND 189
3270 GOTO 3820
3280 DEF PROCSUM3ON
3290 C=C OR 132
3300 GOTO 3820
3310 DEF PROCSUM3OFF
3320 C=C AND 187
3330 GOTO 3820
3340 DEF PROCSUM4ON
3350 C=C OR 136
3360 GOTO 3820
3370 DEF PROCSUM4OFF
3380 C=C AND 183
3390 GOTO 3820
3400 DEF PROCC1ON
3410 C=C OR 144
3420 GOTO 3820
3430 DEF PROCC1OFF
3440 C=C AND 175
3450 GOTO 3820
3460 DEF PROCC2ON
3470 C=C OR 160
3480 GOTO 3820
3490 DEF PROCC2OFF
3500 C=C AND 159
```

```
3510 GOTO 3620
3520 DEF PROCC3ON
3530 D=D OR 193
3540 GOTO 3840
3550 DEF PROCC3OFF
3560 D=D AND 254
3570 GOTO 3840
3580 DEF PROCC4ON
3590 D=D OR 194
3600 GOTO 3840
3610 DEF PROCC4OFF
3620 D=D AND 253
3630 GOTO3840
3640 DEF PROCCLK
3650 V=?&FE60
3660 ?&FE60=(V AND 254)
3670 FOR DEL=1 TO 10
3680 NEXT DEL
3690 ?&FE60=V
3700 ENDPROC
3710 DEF PROCSLOCLK
3720 V=?&FE60
3730 ?&FE60=(V AND 253)
3740 FOR DEL=1 TO 10
3750 NEXT DEL
3760 ?&FE60=V
3770 ENDPROC
3780 ?&FE61=A
3790 GOTO 3850
3800 ?&FE61=B
3810 GOTO 3850
3820 ?&FE61=C
3830 GOTO 3850
3840 ?&FE61=D
3850 ENDPROC
3855
5000 DEF PROCDELAY
5010 FOR DELAY=1 TO 60
5020 NEXT DELAY
5030 ENDPROC
```

Appendix D

Published Papers

D.1 Papers Included in the Appendix

This appendix contains the most important papers published by the author which are relevant to the research in this thesis. Several other papers have been published and presented at various forums, but are similar in content to those listed below and have not been included in this appendix.

The following papers are included in chronological order:

Evans R A (1985), *A Self Organising, Fault Tolerant, 2-Dimensional Array*, Proc. VLSI-85, Tokyo, Japan, ed E Hoebst, North Holland 1986, pp 239-248.

This is the first publication of the self-organising technique and describes the *WINNER* algorithm applied to one dimension of an array only. Circuitry for automatically entering and retrieving data from the functional rows of the array is described.

Evans R A, McCanny J V and Wood K W (1985), *Wafer Scale Integration Based on Self-Organisation*, Proc. Workshop on Wafer Scale Integration, Southampton, ed C Jesshope and W Moore, Adam Hilger, 1986, pp 101-112.

This paper extends the *WINNER* concept and shows how it can be applied to both dimensions of an array.

Evans R A and McWhirter J G (1987), *A Hierarchical Testing Strategy for Self-organising Fault-tolerant Arrays*, in 'Systolic Arrays', eds Moore, McCabe and Urquhart, Adam Hilger (Bristol) UK, pp 229-238.

Introduces the scan-path technique for testing the control circuity in *WINNER*.

D.2 Other Publications

The following publications are not included in this appendix since they are strongly based on chapters of this thesis:

Evans R A (1989), *Wafer Scale Integration* in 'Design and Test Techniques for VLSI and WSI Circuits', R E Massara ed., Peter Peregrinus Ltd, London, to be published 1989.

This is a review of wafer scale integration research based on chapter 4 to be published in book form. In addition it contains a section describing the motivation behind the research into WSI.

Evans R A (1989), *Self-organising Arrays for Wafer Scale Integration* in 'Design and Test Techniques for VLSI and WSI Circuits', R E Massara ed., Peter Peregrinus Ltd, London, to be published 1989.

This is to be published in the same book as above, and is a detailed description of the *WINNER* algorithm, including some details about its performance.

A SELF-ORGANISING FAULT-TOLERANT, 2-DIMENSIONAL ARRAY

Richard A. Evans

Royal Signals and Radar Establishment
St Andrew's Road, Malvern, Worcs,
WR14 3PS, England

A self-organising, fault tolerant algorithm for generating 2-dimensional arrays is described. The algorithm is completely stable for all processor fault distributions and requires no external control. Any required degree of fault tolerance may be introduced by incorporating additional rows into the array, with an overhead of about 20 gates/processor. The technique appears to be very suitable for use in the area of Wafer Scale Integration and high reliability systems.

1. INTRODUCTION

The past few years have seen a dramatic increase in interest in fault tolerant techniques suitable for use with hardware designs. This increase can be largely attributed to two main factors. Firstly, there is broad agreement in the VLSI community that larger chips will be required in the future and in order to keep the cost of these chips at an economic level, manufacturers will need to consider employing fault tolerant techniques to increase the very low yields expected at these chip sizes. Secondly, widespread importance is now being placed on regular architectures consisting of arrays of identical processing elements. It is well known that these arrays can offer high computation rates by exploiting parallel processing and pipelining as well as simplifiying the design process. In addition, the regularity allows redundant elements to be incorporated into the arrays in a very simple manner and these in turn can be used to replace any faulty elements which occur in the active part of the array.

Regular arrays of processing elements can be considered in two categories. The first category includes arrays constructed from simple processing elements each of which contains a small number of gates. An example of such an architecture is the bit-level systolic array [1], where each element contains a single full adder and a few latches. An array containing elements of this type would normally be implemented as a single chip and for this reason it has been termed a 'chip level array'.

The second category includes arrays in which the individual elements are more complex. Examples are array processing computers like the DAP [2] and GRID [3], systolic arrays of the type proposed by Kung [4], and arrays of Transputers [5]. With these arrays, each element is likely to require a complete chip for its implementation, while the whole array may be considered suitable for implementation as a Wafer Scale system. These arrays have been termed 'system level arrays'.

Copyright © Controller HMSO, London, 1985.

The fault tolerant strategy used with a particular array will depend upon the category into which the array falls. At the chip level, a recently published technique [6], involves the use of redundant rows of processors which replace complete rows of the array containing one or more faults. Each row containing a fault is bypassed and a spare is switched in to replace it. This technique is effective when used in chip level arrays because each row of the array is a fairly simple circuit and the amount of good circuitry discarded by switching out a faulty row is small. In addition the switching operation can be performed by some very simple additional logic.

At the system level, the complexity of each cell means that it is inefficient to discard a whole row of cells if just one of its members is faulty. Furthermore, if a fixed maximum percentage overhead of extra control circuitry for performing the fault tolerant operation is permitted, then it will be possible to include a larger absolute quantity of logic per processor at the system level than at the chip level. This presents the opportunity of employing a more intelligent fault tolerant strategy.

This paper addresses the problem of applying fault tolerance to system level arrays. As our key concept we propose a method by which an array of processors can be given the ability to automatically organise itself in such a way as to construct a functional 2-dimensional array from a given 2-dimensional array containing a number of faulty processors. Redundancy is introduced in the form of additional rows of processors, the number of which can be chosen to give the degree of fault tolerance required. The approach appears to be suitable for use both with arrays of interconnected chips or at the WSI level.

In section 2 we present the proposed algorithm and a descibe a practical implementation of it in Section 3. Section 4 gives details of the theoretical and simulated performances for various yield characteristics, while a method of improving the user interface to a configured array is presented in Section 5. Finally, in Section 6 we discuss the merits of the technique and consider ways in which it might be improved.

2. THE ALGORITHM

The aim of this algorithm is to construct a 2-dimensional array of orthogonally interconnected processors with connections in the x and y directions. It is assumed that some mechanism exists whereby each processor

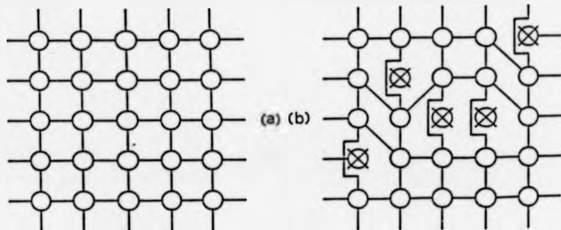


Figure 1. (a) Perfect array, (b) Configured imperfect array.

is able to indicate whether or not it is faulty. In practice this might be achieved by some form of self-testing scheme, or perhaps by an externally applied testing strategy.

Figure 1(a) illustates the type of array being considered. The array shown has a perfect interconnection pattern since there are no faulty processors in the array. Figure 1(b) shows a similar array but which now contains a number of faulty processors each indicated by a cross. It can be seen that an array which functions as an orthogonal array (termed a functionally orthogonal array) can still be constructed by using the interconnections shown. The interconnections in the horizontal direction have been bent by allowing processors to communicate with their diagonal neighbours if necessary and this enables rows of functional processors to avoid faulty processors. It is obvious that this functional array will be smaller than the corresponding perfectly connected array, but it should be noted that the x dimension of the functional array is identical to that of the given array, while the dimension in the y direction will depend upon the fault distribution of the array. This is characteristic of the algorithm to be described.

An interconnection pattern such as that described above could be achieved by laser programming or by electrical fuse blowing. However, the method proposed here relies upon logical configuration which will occur automatically when the array is switched on. In order to achieve this it is necessary to associate some additional control circuitry with each processor in the array. The combination of a processor and its control circuitry will be called a 'cell'. A cell interconnection pattern suitable for use with the algorithm is illustrated in figure 2. It should be noted that although an orthogonally interconnected array of functional processors is to be constructed, a more complex cell interconnection scheme is required in order that faulty cells can be avoided. The cell illustrated has a single connection in the north-south direction which is identical to that of the processor. However in the east-west direction, sufficient communication channels have been provided to allow a cell to communicate Westwards with its NW, W or SW neighbours, and Eastwards with its NE, E or SE neighbours.

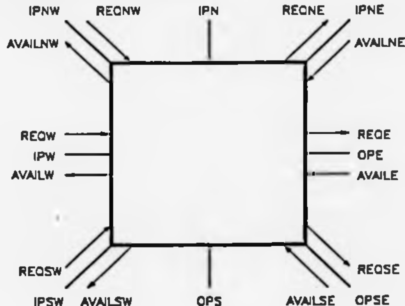


Figure 2. Cell Input/Output requirements.

The function of each cell is to establish communication channels between its internal processor and the processor of a neighbouring cell in the columns to its left and right, so that, in a global context, fault-free processors are connected to form a number of functional rows which together make up the required functional array. Faulty or unused processors are bypassed in the north-south direction by the control circuitry which effectively eliminates them from the functional array.

Each cell has a number of control inputs in addition to the processor communication channels. Referring to figure 2 it can be seen that signals labelled REQNW, REQW and REQSW enter from the NW, W and SW directions respectively. These signals are 'request' signals, and similar signals leave the cell in the directions of NE, E and SE as shown. Each cell also has control signals labelled AVAILNE, AVAILE and AVAILSE entering from the NE, E and SE directions respectively. These are 'availability' signals, and corresponding signals leave the cell in the directions of NW, W and SW.

If a cell A outputs a true request signal to some other cell B it means that cell A wishes to set up a communication channel between its processor and the processor in B. If such a communication channel becomes set up then A is said to have been 'connected' to B. If a cell A outputs a true availability signal to another cell B it means that cell A contains a processor which is available for connection if requested by cell B. The manner in which these signal are generated by a cell forms the heart of the algorithm.

Availability Signals

A cell generates its availability output signals according to the following rules:

(1) A cell can only output a true availability signal if it contains a processor which is fault-free (ie the self-test shows it to be functional), and at least one true availability signal has been received from its NE, E or SE neighbours.

(2) If (1) is satisfied, then the following priority system operates to decide in which directions to send true availability signals:

Request Inputs			Availability Outputs		
REQNW	REQW	REQSW	AVAILNW	AVAILW	AVAILSW
TRUE	X	X	TRUE	FALSE	FALSE
FALSE	TRUE	X	TRUE	TRUE	FALSE
FALSE	FALSE	TRUE	TRUE	TRUE	TRUE
FALSE	FALSE	FALSE	TRUE	TRUE	TRUE

X = Any Value

This scheme allows a priority of connections to be established so that a request from the NW has highest priority, and requests from the W and SW have successively lower priorities. Such a scheme is required in an iterative system to resolve problems such as a cell A becoming connected to a cell to the SW of A in advance of receiving a request from a cell to the NW of A.

The second request has higher priority and must be able to override the SW connection and establish its own connection in its place. In addition, the scheme causes a FALSE availability signal to be output to cells which have no chance of obtaining a connection to A, ie if A is already connected to another cell with some priority, a cell with lower priority cannot become connected to A.

The first rule gives the cell a global look-ahead capability even though each cell is capable only of local communication. Information is passed between cells from east to west about the availability of other cells. This allows a cell A to prohibit another cell from connecting to it if A either contains a faulty processor or would be part of a dead-end route, ie a route that would not be able to be completed due to some blockage later. The scheme allows information about blockages to be transmitted from right to left to all the relevant processors, which then decide upon some appropriate avoiding action.

Request Signals

Request signals are output from a cell according to a different set of rules:

(1) A cell can only output a true request signal if its processor is fault-free and at least one request has been received from one of its NW, W or SW neighbours.

(2) If (1) is satisfied then the cell outputs a single true request value to one of its NE, E and SE neighbours according to the priority tabulated below:

Input Availability			Output Request		
AVAILNE	AVAILE	AVAILSE	REQNE	REQE	REQSE
TRUE	X	X	TRUE	FALSE	FALSE
FALSE	TRUE	X	FALSE	TRUE	FALSE
FALSE	FALSE	TRUE	FALSE	FALSE	TRUE
FALSE	FALSE	FALSE	FALSE	FALSE	FALSE

X = Any value

The rules ensure that only one request signal is output from any cell, which in turn ensures that a cell can never accidentally become connected to more than one neighbouring cell in the Easterly and Westerly directions.

The availability and request signals together provide the cells with all the information they need about their surroundings in order to be able to form functional rows of interconnected processors. The priority system for sending and receiving request and availability signals ensures that stable functional rows are established from west to east and from north to south starting in the top left hand corner of the array. The priority system also ensures that each row formed is as close as possible to the northern edge of the array.

It is possible for a processor to be omitted from the functional array for several different reasons. These can be best seen by reference to figure 3. Cells marked with a cross are obviously omitted because they are faulty.

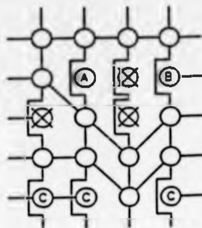


Figure 3. Some good processors are not used.

However, although cells A and B are fault-free, they have been omitted from the functional array because of the presence of faults in nearby cells as shown, which cause a shadowing effect. The control circuitry in cell A will detect the fact that all of its NE, E and SE neighbours are unavailable and will declare itself to be unavailable as a result. Cell B has been omitted because it cannot be requested by its NW, W or SW neighbours, since they are either faulty or already connected to another cell with higher priority. Cells labelled C are omitted because there are insufficient available cells to allow further complete functional rows to be formed.

All cells which are either faulty or unused for any reason are controlled to activate a north-south bypass which simply renders the cells transparent in the vertical direction.

3. CELL IMPLEMENTATION

Control ciruitry suitable for use with processors having one input or output wire in each of the N, S, E, and W directions has been designed and introduces an overhead of approximately 20 two-input gates per processor. For processors with more than one wire in these directions it will be necessary to add some extra gates for each extra wire. A 2-dimensional array of these cells has been described at the gate level in the hardware description language, ELLA [7], and the operation of the array has been simulated for a number of different processor fault distributions. A correctly configured circuit has been produced each time.

In a practical implementation of this technique it is possible to visualise the whole array as having two sections. One section comprises an underlying array of control circuitry which is capable of establishing communication channels between appropriate neighbours to generate a functionally orthogonal array as described. The second section is an array of processors containing a number of faulty devices which can be considered to be overlaid on the array of control circuitry which then forms interconnections between processors as appropriate. Since the control overhead in array is only about 20 gates per processor the probability of a fault occuring in the control circuitry is likely to be acceptably low. However if desired it should be possible to use techniques such as triplication of the control circuitry in order to reduce this probability even further.

4. PERFORMANCE

In order to try to establish the fault tolerant charateristics of the technique theoretically a model of the algorithm was used. In the model we have assumed that all the control circuitry operates correctly and that faults only occur in the processors. In addition we have assumed that the distribution of faults over the array is entirely random and have adopted the simplistic view that each fault acts independently of other faults in influencing the number of functional rows which are generated.

An important feature of the fault distribution is the maximum number of faulty processors occurring in any one column of the array since it is primarily this value which limits the number of processors available for forming functional rows. The calculations give the factor (called the Redundancy Factor), by which the number of rows must be increased in order to achieve a functional array of x by y processors compared with a fault free array of x by y processors. In other words:

Actual no. of rows needed to generate y functional rows = Redundancy Factor * y

The first order theoretical performance is plotted in figure 4.

In order to assess the validity of the model the algorithm was described and simulated in Algol. A number of simulations were performed on an array with 10 rows and 10 columns and random distributions of faults. An estimate of the redundancy factor was then derived for different yield values by averaging the results from 20 simulations with independent fault distributions. The results by calculation and simulation are presented in figure 4. It can be seen that both curves are similar in shape and that there is good correlation at yields above about 60%. However, the simulation results indicate that the redundancy factor rises much earlier than predicted. This is thought to be due to second order effects becoming significant at yields below about 60%. The most dominant effect is likely to be that in the presence of a large number of faults, some of the functional cells will become inaccessible because they are partially or completely surrounded by faulty cells. This will reduce the effective yield of the array and tend to move the theoretical curve to the right.

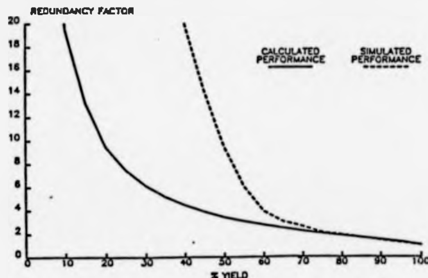


Figure 4. Theoretical and simulated performances.

5. TRANSPARENT USER INTERFACE

The technique so far described seems to be potentially useful, but still leaves much to be desired from the users point of view. The user needs to know, for example, where functional rows of processors start and end in order to apply inputs and receive outputs from the array. In the vertical direction this is not such a problem because the functional array is the same width as the given array.

In the horizontal direction, however, true availability signals emerging from the W side of the array mark the start of each functional row, while on the E side, true request signals mark the ends of functional rows. These signals can be used in conjunction with some simple extra circuitry to make the fault-tolerant array appear like a perfect array to the user, who need not be aware that he using a physically imperfect array.

This task can be performed using an input array of cells and an output array of cells whose circuitry is illustrated in figures 5 and 6. Figure 5 shows a single cell for each of the input and output arrays, while figure 6 illustrates the cell interconnections required to form the arrays. It should be noted that the cells are mirror images of each other with the input cell having an extra wire passing though it. Each of the input and output arrays has as many rows as there are physical rows in the self-configuring array, and

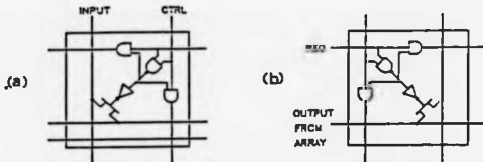


Figure 5. Cells for: (a) Input array, (b) Output array.

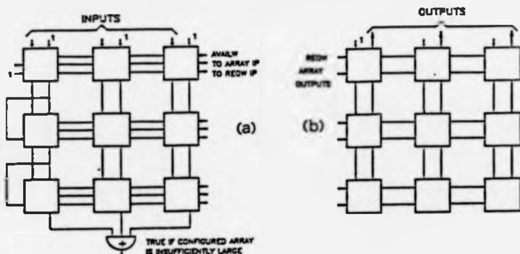


Figure 6. (a) Input and (b) Ouput Interface arrays.

as many columns as there are horizontal inputs or outputs to the configuring array. The input array operates as follows. The CTRL inputs at the top of the array are set to TRUE and signals are applied to signal inputs. Each CTRL signal passes down its respective column until it encounters a cell which has a TRUE availability signal arriving from the East. This cell is then controlled to connect its signal input to its horizontal signal line which in turn is connected to the main array. The cell then outputs FALSE CTRL and AVAILW values. This prevents any other signal inputs from becoming connected to the same input of the main array, and also prevents the signal input just connected to the main array from becoming connected to any other input of the main array. In this way, the rightmost signal input applied to the input array will be connected to the first available functional row starting from the top of the array, with other inputs becoming connected to successively lower functional rows. It should be noted that the NW, SW, NE and SE inputs to cells on the periphery of the array are connected to a FALSE value. This provides the necessary boundaries within which the configuration algorithm is to operate.

It is important to note that some special action is necessary if the number of functional rows generated by the self-configuring circuitry exceeds the number of signal inputs. In this case we must ensure that the extra functional rows are bypassed in the north-south direction. This can be simply achieved by feeding the CTRL output from each cell in the left hand column into the REQU input of the next row of the main array as shown. This ensures that TRUE request signal are applied to all functional rows except those which are in excess of the required number.

The output array operates in a similar manner to the input array, except that it is now CTRLed by the REQUEST signals emerging from the main array. Outputs appear at the top of the array with the output from the topmost functional row being on the left.

An interesting feature of either the input or output arrays is that they can also provide information about whether sufficient functional rows have been found for the number of signal inputs which have been applied. Such an indication can be generated by ORing the CTRL outputs emerging from the bottom of either array. If sufficient functional rows are available, all the CTRL outputs should be FALSE. However, if some signal inputs do not have a row to connect to, one or more of the CTRL outputs will still be TRUE.

6. DISCUSSION

In this section we consider the merits of the technique and discuss the validity of the assumptions which have been made.

The technique possesses a number of attractive properties. These include the simplicity of the control circuitry, stability for all processor fault distributions, and the fact that the algorithm requires no global control. In addition the technique is applicable to any orthogonally connected array regardless of size.

A number of assumptions have been made. Firstly it has been assumed that each processor can test itself. This is not unreasonable as there is currently much interest in the area of self-test. However it will be necessary to ensure that a processor does not incorrectly declare itself to be functional due to some fault in the test circuitry. Secondly, it has been assumed that the control circuitry in each cell is fault-free. This is not a totally valid assumption but the probability of it being true can be increased significantly by using other fault tolerance techniques, for example triplication with

voting, or by relaxing the design rules for critical parts of the circuit. The assumption that faulty processors will be distributed entirely randomly across the array may not always be valid. In the case of a wafer, it is likely that faults will occur in clusters; in addition, a processor close to the array edge will have a higher probability of failure than one nearer to the middle of the array. It may be possible, however, to overcome this to some extent by, for example, discarding the outer portion of the wafer.

The algorithm could find application in any system which involves the use of arrays of identical processors. It could be used to enhance the reliability of a system, for example a Distributed Array Processor [5], or to improve the yield of a Wafer Scale Integrated Circuit. The minimum complexity of the processors will be governed by the acceptable overhead presented by the control circuitry. In addition to its use for generating functional 2-dimensional arrays, the technique could also be used to construct a linear chain of processors by joining the ends of the functional rows. In this case the processor bypass circuitry would not be required.

REFERENCES

[1] Evans R A, et al, 'A CMOS Implementation of a Systolic Convolver Chip', Proc VLSI-83, Trondheim, Norway, August 1983.

[2] Hockney R W and Jesshope C R, 'Parallel Computers', Adam Hilger Ltd., Bristol, pp 178-192.

[3] Corry A G, et al, 'Image Processing with VLSI', Proc NATO ASI on Impact of Processing Techniques on Communications, Chateau de Bonas, France, July 11-22, 1983.

[4] Kung H T and Leiserson C R, 'Algorithms for VLSI Systems', Section 8.3 of 'Introduction to VLSI Systems' by Mead C and Conway L, Addison Wesley, 1980

[5] 'INMOS Preliminary Data Sheet IMS T424 Transputer', INMOS Ltd, Nov 1984.

[6] McCanny J V and McWhirter J G, 'Yield Enhancement of Bit-Level Systolic Array Chips using Fault Tolerance Techniques', Electronics Letters, 7 July 1983, Vol 19, No 14, pp 525-527.

[7] Morison J D, et al, 'ELLA: A Hardware Description Language', Proc IEEE, ICCC 82, Sept 1982, pp 604-607.

3.5 WAFER SCALE INTEGRATION BASED ON SELF-ORGANISATION

R A Evans, J V McCanny and K W Wood

INTRODUCTION

Advances in VLSI technology have led to increasing interest in two-dimensional processor array architectures as a means of implementing hardware systems which are required for the high speed computation of highly structured operations. Such applications are encountered in real-time digital signal and image processing and in scientific computation (Reddaway 1979, Robinson and Moore 1982, Duff 1978, Kung and Leiserson 1980). Given the current trend towards wafer scale integration (WSI) (McDonald *et al* 1984, Moore 1986), it is important to consider how such architectures could benefit from developments in this type of technology, particularly as they exhibit a number of features which are attractive from a WSI point of view. First, their highly regular nature should make such systems easier to design than ones based on random logic. Secondly, their strong dependence on nearest neighbour connections should help avoid problems associated with propagation delays on long random interconnects. Such problems appear to have been the major reasons why Trilogy's recent WSI venture did not result in the production of commercial devices (McDonald *et al* 1984).

A number of techniques have now been proposed and/or demonstrated which are applicable to two-dimensional processor arrays. Broadly speaking these can be divided into two main classes: (*a*) those which required some form of post-processing to be done to the wafer, such as discretionary wiring or the use of lasers to make or break electrical connections (Raffel *et al* 1984, Petritz 1967); and (*b*) those which involve reconfiguration by electronic switches which are usually addressed from the edge of the wafer (Hedlund and Snyder 1984, Katevenis and Blatt 1985).

The main advantage of the second class of techniques over the first is that reconfiguration can be carried out during normal circuit operation and this allows further faults to be avoided as they develop during use. The scope for doing this using post-fabrication techniques is extremely limited. However,

Copyright © Controller HMSO, London, 1986.

the use of programmable switches and their associated mesh of control buses (which in general must be designed to have a high probability of working) represents a considerable overhead in terms of silicon area and power consumption by comparison with techniques such as laser welding.

In this section we examine fault tolerance in two-dimensional processor arrays and present what we believe to be a novel solution to the problem. The general approach which we will describe could be classified under (*b*) above, in that it is based on electronic switching. However, it eliminates the need for any global control and occurs automatically when the array is switched on. The technique utilises the cellular properties of arrays and is applicable to systems such as systolic arrays which have nearest neighbour interconnections only. Each cell within the array is given some intelligence in the form of a small amount of additional circuitry. This enables each processing element independently to make local decisions about how it should be connected to neighbouring elements, taking into account its own functionality, the functionality of its neighbours and the connection priorities to these neighbours which are defined in specific algorithms. The effects of these local decisions propagate throughout the array and manifest themselves globally as a complete self-organisation of the functional processing elements into a correctly interconnected functional two-dimensional processor array.

A number of algorithms incorporating these basic concepts can be derived. For our present purposes we concentrate our attention on two related algorithms which illustrate the basic technique. The first scheme is the simpler of the two: the second method to be described is of a more general nature. The relative merits of the two approaches are considered, along with a number of other important issues such as practical implementation, and ability to cope with various fault distributions. The major conclusions which can be drawn from the work are presented at the end.

ALGORITHM NUMBER 1

The aim of both algorithms to be described in this section is to construct an orthogonally interconnected two-dimensional array of processors of the type shown in figure 3.5.1(*a*) from an array of processing elements, some of which may be faulty. Figure 3.5.1(*b*) shows an example of an array which has been connected in such a manner, with faulty processors each indicated by a cross. The interconnections in the horizontal direction have been altered in the vicinity of faulty elements by allowing processors to communicate with their diagonal neighbours and this enables rows of functional processors to avoid faulty processors. It is obvious that the functional array generated in this way will be smaller than the corresponding perfect array due to the presence of the faulty elements but it should be noted that for the wiring

scheme shown in figure 3.5.1(*b*) the x dimension of the functional array is identical to that of the given array, while the y dimension depends on the number of faults which occur in each column of the array.

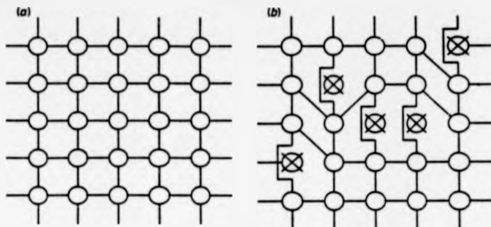


Figure 3.5.1 (*a*) Perfect array, (*b*) array with faults avoided.

An array of processors containing faulty elements can be given the ability to organise itself into a structure similar to that shown in figure 3.5.1(*b*) if the following assumptions are made: (i) that each processor contains some form of self-testing circuitry which allows it to indicate whether it is working, and (ii) that all connections in the vertical direction can be designed so that they are fault-free and are organised so that initially, at switch-on, all processors are bypassed. Bypass connections around a processor are only removed to allow a cell to become part of the functional array if the cell itself is functional and is contained within a functional row.

The method proposed for fault tolerance is assumed to occur automatically when the array is switched on and is a totally asynchronous technique. The decisions which cells make about their connections to neighbours depend not only on whether a neighbour is faulty but also on the decisions being made by those neighbours about their own environment. In the early stages of self-configuration the situation may be highly dynamic with cells forming and relinquishing connections to other cells as a result of being overridden by higher priority decisions which have been made at other localities and have rippled through the array. Connections may in fact experience a number of iterations of this type but will always settle into a self-consistent, stable state.

A basic cell with the required self-healing capability is shown in figure 3.5.2. It should be noted that although the N–S connections are similar to those required in a non-fault-tolerant circuit, extra channels have been provided on the eastern and western sides which allow the cell to communicate with neighbours to the NW, W and SW, and the NE, E and SE respectively. Each cell also has a number of control inputs in addition to the

processor communication channels. These are indicated in figure 3.5.2 as request (REQ) and availability (AVAIL) signals.

If a cell A outputs a TRUE request signal to some other cell B it means that cell A wishes to set up a communication channel between its processor and the processor in B. If such a communication channel becomes set up then A is said to have been 'connected' to B. If a cell A outputs a TRUE availability signal to another cell B it means that cell A contains a processor which is available for connection if requested by cell B. The manner in which these signals are generated by a cell forms the heart of the algorithm.

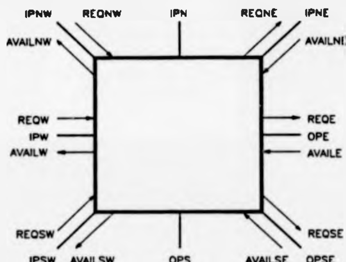


Figure 3.5.2 Basic cell having self-healing capability.

Availability signals

A cell generates availability output signals according to the following rules:

(i) A cell can only output a TRUE availability signal if it contains a processor which is fault-free (i.e. the self-test shows it to be functional), and at least one TRUE availability signal has been received from its NE, E or SE neighbours.

(ii) If (i) is satisfied, then the priority system of table 3.5.1 operates to decide in which directions to send TRUE availability signals.

Table 3.5.1

Request inputs			Availability outputs		
REQNW	REQW	REQSW	AVAILNW	AVAILW	AVAILSW
TRUE	×	×	TRUE	FALSE	FALSE
FALSE	TRUE	×	TRUE	TRUE	FALSE
FALSE	FALSE	TRUE	TRUE	TRUE	TRUE
FALSE	FALSE	FALSE	TRUE	TRUE	TRUE

This scheme allows a priority of connections to be established so that a request from the NW has highest priority, and requests from the W and SW have successively lower priorities. Such a scheme is required in an iterative system to ensure that a stable solution is reached. The scheme causes cells to output FALSE availability signals to neighbouring cells if they have no chance of obtaining a connection. This might occur for example when a higher priority connection has already been established.

The first rule gives the cell a global look-ahead capability even though each cell is capable only of local communication. This enables clustered faults to be avoided in the following way. Information is passed between cells from east to west about the availability of other cells. This allows a cell A to prohibit another cell from connecting to it if A either contains a faulty processor or would be part of a dead-end route, i.e. a route that would not be able to be completed due to some blockage later. Such a dead-end route could occur, for example, if three vertically adjacent processors were faulty. In this case a functional processor to the left of the centre faulty processor would find that all of its possible connections to neighbours are unavailable. The functional processor would then declare itself to be unavailable. The scheme allows information about blockages to be transmitted from right to left to all the relevant processors, which then decide upon some appropriate avoiding action.

Request signals

Request signals are output from a cell according to a different set of rules:

(i) A cell can only output a TRUE request signal if its processor is fault-free and at least one request has been received from one of its NW, W or SW neighbours.

(ii) If (i) is satisfied then the cell outputs a single TRUE request value to one of its NE, E and SE neighbours according to the priority tabulated in table 3.5.2.

These rules ensure that only one request signal is output from any cell, which in turn ensures that a cell can never accidentally become connected to more than one neighbouring cell in the easterly and westerly directions.

Table 3.5.2

Input availability			Output request		
AVAILNE	AVAILE	AVAILSE	REQNE	REQE	REQSE
TRUE	×	×	TRUE	FALSE	FALSE
FALSE	TRUE	×	FALSE	TRUE	FALSE
FALSE	FALSE	TRUE	FALSE	FALSE	TRUE
FALSE	FALSE	FALSE	FALSE	FALSE	FALSE

The availability and request signals together provide the cells with all the information they need about their surroundings in order to be able to form functional rows of interconnected processors. The priority system for sending and receiving request and availability signals ensures that stable functional rows are established from west to east and from north to south starting in the top left-hand corner of the array. The priority system also ensures that each row formed is as close as possible to the northern edge of the array, thus maximising the number of rows generated.

ALGORITHM NUMBER 2

The above approach makes the basic assumption that it is always possible to bypass faulty cells in the vertical direction. However, when we started this work our basic philosophy was that the configured array should completely avoid all faulty processors. In this section we will show how the technique of algorithm 1 can be extended to encompass this philosophy by configuring both the rows and the columns. To see how this is done the reader is referred to figure 3.5.3.

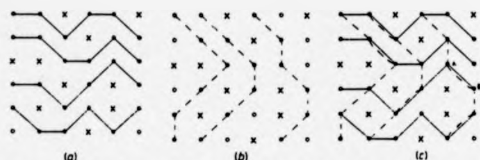


Figure 3.5.3 (*a*) Functional rows, (*b*) functional columns, (*c*) functional array.

Figure 3.5.3(*a*) illustrates a typical array which has been configured in the horizontal direction using algorithm 1. Functional rows have been generated which each contain a number of working processors equal to the width of the original array. Figure 3.5.3(*b*) illustrates the same array which has now been configured in the vertical direction using an identical algorithm operating vertically. Here, columns are constructed which keep as close as possible to the left-hand side of the array and each functional column contains cells equal to the height of the original array.

Since the configurations generated by the algorithm consist of full width rows and full height columns, the superposition of the rows and the columns must generate an array of functional processors at the points of intersection of each row with each column. In addition, the processors within the array region will have orthogonal interconnections. The superposition of the rows

and columns is shown in figure 3.5.3(c), where the processors forming the final array are indicated in black. Some processors have either horizontal or vertical connections but not both. These are controlled to act as bypasses in the direction in which they have connections. The size of the functional array is determined by the number of functional rows and columns generated. If p is the number of functional rows and q the number of functional columns, then superposition will generate a functional array of dimensions $p \times q$. A cell suitable for use with this algorithm would have extra communication channels and control signal paths for both the horizontal and vertical directions.

There are in fact two types of undesirable condition which can occur when the rows and columns are superimposed. These have been called 'double site' and 'crossover' conditions and must be handled separately if a correct array is to be produced. The example illustrated here contains several double sites.

Double Site Condition

Referring to figure 3.5.4(a) we see that there are two processors (A and B) belonging to the same functional row which both occur in the same functional column (or vice-versa). The effect of this is that there are two processor sites, A and B, where only one is required. The problem can be overcome by instructing one of the processors to act as a bypass in both the horizontal and vertical directions. In our implementation we always instruct the upper processor to become the bypass. The rule for doing this is as follows:

> If a cell finds that it is requesting to be connected to a cell to its SW or SE for both its rows and its columns, it will become a bypass for its horizontal and vertical connections.

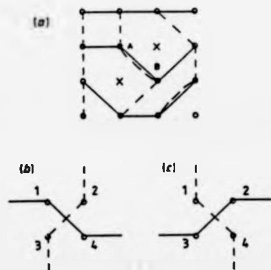


Figure 3.5.4 (a) Double site condition, (b), (c) crossovers.

At first sight it may appear that since the process of avoiding double sites requires functional processors to be discarded the functional array size might be reduced as a result. However, these processors could not have formed part of the functional array anyway and discarding them does not affect the array size of $p \times q$.

Crossovers

This problem occurs when a row and a column intersect each other at a point other than a processor site. Crossovers can occur in two distinct ways as shown in figures 3.5.4(*b*) and (*c*). Unlike the double site condition which can be overcome without altering the configured rows and columns, the solution to the crossover condition requires either the row or the column containing the crossover to be physically altered so that the superposition crossover does not occur. Alteration of the row or column may of course produce effects which propagate throughout the array until a new stable configuration is achieved.

We have devised a technique which avoids the crossover condition. It requires the use of an extra bit in both the horizontal and vertical directions. These extra bits allow the rows to be generated as before, but restrict the generation of columns to sites which will not cause crossovers. It can be seen from figure 3.5.4(*b*) that the crossover could be avoided if cell 3 was made 'unavailable' to cell 2. In figure 3.5.4(*c*), the crossover could be avoided if cell 4 was made 'unavailable' to cell 1. In the first case, this can be achieved by using an extra bit which propagates between cells in the north–south direction. The bit indicates whether or not a cell is outputting a row request in the SE direction and if so it causes the column AVAILNE signal in the cell below to be inhibited. The second crossover case may be avoided in a similar manner by passing an extra bit from west to east, indicating whether a cell has output a row request to the NE, and if so inhibiting the column AVAILNW in the cell to its right. The cost of this technique is two extra inputs and outputs plus two (A AND NOT B) logic functions to perform the inhibitory action.

We have also investigated the possibility of avoiding crossovers without using any extra bits by exploiting features of the availability signals. This is in fact possible but can unfortunately become unstable for certain fault distributions. For this reason the technique is not described in detail here and would not be recommended for use in practice.

DISCUSSION

A number of important questions arise concerning the implementation and application of the two schemes described. In both approaches the basic

assumption is made that each processor has the ability to test itself. This is felt to be a reasonable assumption given the current interest in the area of self-test and the fact that a number of chips are now available which possess this capability. It is also important to note that although the algorithms described in this paper have been presented with wafer scale integration in mind they are equally applicable to high availability and high reliability systems. For example, a major application of the techniques may be to circuits such as multi-processors on a hybrid or printed circuit board. The basic assumption of fault-free bypass circuitry in the vertical direction which is implicit in the first algorithm should be reasonably easy to achieve for this type of application.

Control circuitry suitable for use with processors with single input and output lines in each of the N, S, E and W directions has been designed for both methods. In the first case this introduces an overhead of approximately 20 two-input gates per processsor whilst in the case of the second method the corresponding figure is roughly doubled. For systems in which processors are connected via multi-bit buses, additional circuitry is required for each wire in order to allow the whole bus to be routed to the appropriate neighbour. Generally speaking, the overheads required are relatively small but the approach is obviously best suited to systems in which interprocessor communication is by serial links. Systems built from the INMOS Transputer, for example, are therefore seen as ideal candidates for the methods described (INMOS 1984).

Both algorithms have been validated using the hardware description language ELLA (Morison *et al* 1982). This has allowed the techniques to be described and simulated at the gate level for a number of different fault distributions. In all cases correctly configured arrays were produced.

In a practical implementation of this technique it is possible to visualise the whole array as having two sections. One section comprises an underlying asynchronous network of control circuitry which is capable of establishing communication channels between appropriate neighbours to generate a functionally orthogonal array as described. The second section is an array of processors containing a number of faulty elements which can be considered to be overlaid on the array of control circuitry which then forms connections between processors as appropriate. Since the control overhead in an array is only a few tens of gates per processor the probability of a fault occurring in the control circuitry is likely to be acceptably low. However, if desired, it should be possible to use techniques such as triplication of the control circuitry in order to reduce this probability even further.

An interesting application of our approach is in self-timed systems such as wavefront array processors (Kung 1982), where the only global signals required are power rails. Such a circuit would consist of a collection of totally autonomous cells each with the capability of forming links with its neighbours to generate a functional two-dimensional array and each with the

independent capability of controlling the timing of information between itself and its neighbours.

It is important to ascertain the ability of the two algorithms described to cope with various fault distributions. Models of the algorithms were therefore written in ALGOL and each was simulated for a 10 × 10 array of cells with random distributions of faults. A number of different simulations were carried out for arrays with processor yields of between 50% and 100%. Then by averaging the results obtained at each yield value we were able to estimate the 'overhead factor', which for a given target array size and overall processor yield indicates the factor by which the number of cells in the target array must be multiplied in order that, on average, it will be possible to form the target array. These results are presented in figure 3.5.5. The shapes of both curves are similar with the value of the overhead factor initially rising slowly from unity as the cell yield drops from 100% but then rising more rapidly for yields less than 60% and 80% for algorithms 1 and 2 respectively. This is mainly due to the fact that functional cells start to become inaccessible below these values due to being partially or completely surrounded by faulty cells and this reduces the number of functional rows and columns which can be formed.

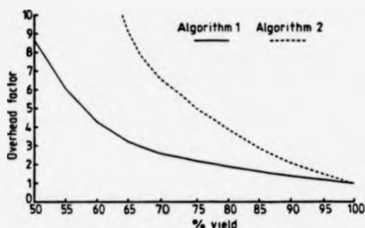


Figure 3.5.5 Algorithm performances.

Since algorithm 2 effectively uses algorithm 1 in both the row and column directions, one would expect the overhead factor for algorithm 2 to be the square of that for algorithm 1. The results of the simulations confirm this and clearly indicate that if bypassing of faulty processors is acceptable then algorithm 1 should be used. An alternative way of expressing the performance of the algorithms is to say that if the initial array has a processor yield of 75% then on average a 'harvest' of 60% of the working processors will be achieved by algorithm 1 and a 25% harvest by algorithm 2.

The discussion up to this point has neglected the important issue of inputs and outputs to the fault-tolerant arrays described. The user of such a circuit

obviously needs to be able to apply his input signals and receive output signals at appropriate points at the extreme ends of functional rows and columns of the array. Connections between a set of input/output ports and the main array can in fact be made by applying the same principles of self-configuration which have been applied to the array itself. As is described in detail elsewhere (Evans 1985), one can make use of the availability and request signals which emerge from the edges of the array to route the inputs and outputs of cells to a set of pads located around the edge of the wafer. With this facility, the array appears as a perfect circuit to the user and he need not be aware that the circuit he is using actually contains a number of faulty processors.

CONCLUSIONS

In this section we have proposed a novel approach to achieving fault tolerance in any processor array using techniques based on self-organisation. For the purposes of illustration we have concentrated our attention on an orthogonally interconnected two-dimensional array and have described two specific algorithms which can be used, each having its own relative merits. However, many variants of the self-organising approach can be developed and can be applied to a range of computational structures, particularly those with regular interconnections. It is hoped to present further discussion on the broader application of the basic concepts in the future.

REFERENCES

Duff M J B 1978 Review of the CLIP4 Image Processing System *Proc. Nat. Comput. Conf.* 1055–1060

Evans R A 1985 A Self Organising, Fault-Tolerant, 2-Dimensional Array *Proc. VLSI-85* (Amsterdam: North-Holland)

Hedlund K and Snyder L 1984 Systolic Architectures—A Wafer Scale Approach *Proc. IEEE Int. Conf. on Computers* 604–610

INMOS Ltd 1984 *INMOS Preliminary Data Sheet IMS T424 Transputer*

Katevenis M G H and Blatt M G 1985 Switch Design for Soft-Configurable WSI Systems *Proc. 1985 VLSI Conf.*

Kung H T and Leiserson C R 1980 Algorithms for VLSI Processor Arrays *Introduction to VLSI Systems* (Reading, Mass.: Addison-Wesley) Ch. 8

Kung S Y *et al* 1982 Wavefront Array Processor: Language Architecture and Applications *IEEE Trans. Comput.* C-31 1054

McDonald J F, Roger E H and Rose K 1984 The Trials of Wafer Scale Integration *IEEE Spectrum* (October) 32–39

Moore W R 1984 A Review of Fault Tolerant Techniques for the Enhancement of Integrated Circuit Yield *GEC J. Res.* 2(1) 1–15

Morison J D *et al* 1982 ELLA: A Hardware Description Language *Proc. IEEE ICCC 82* 604–607

Petritz R I 1967 Current Status of LSI Technology *IEEE J. Solid State Circuits* 2(4) 130–147

Rafel J *et al* 1984 A Wafer Scale Digital Integrator *Proc. IEEE Int. Conf. on Computer Design: VLSI in Computers (ICCD '84)* 121–126

Reddaway S F 1979 The DAP Approach *Infotech State of the Art Report: Supercomputers* 2 311–329 ed C R Jesshope and R W Hockney (Maidenhead: Infotech International Ltd)

Robinson I N and Moore W R 1982 A Parallel Processor Array Architecture and its Implementation in Silico *Proc. IEEE CICC* 41–45

A HIERARCHICAL TEST STRATEGY FOR SELF-ORGANISING FAULT-TOLERANT ARRAYS

R A EVANS and J G McWHIRTER

INTRODUCTION

In the past, increases in the performance of electronic systems have to a large extent been gained as a result of improved device processing. This has provided higher levels of integration together with reduced device propagation delays. The realisation that devices are approaching fundamental performance limits and the fact that for many applications the advances are not keeping pace with the desire for faster processing has been fuelling interest in parallel architectures for a number of years and is now a major driving force behind the increasing interest in Wafer Scale Integration.

The main difference between single chips and wafer scale devices is that Wafer Scale Integration, or WSI for short, requires fault tolerance to be built in as a standard procedure before devices can be fabricated; chips in general do not require fault tolerance although some specialised devices, such as memories, have incorporated fault tolerance for many years. For this reason, many researchers in the WSI field, for example Chevalier and Saucier (1985), Raffel (1985), and Moore and Mahat (1985) have devoted their efforts to developing techniques by which working systems can be generated from systems containing faulty components. Most of the work has focussed on arrays of processors since their regularity allows a global pool of redundant elements to be employed. Each redundant element can in principle be used to replace a faulty cell anywhere in the array provided that the appropriate switching arrangement is incorporated.

Several problems require study. Firstly, one and two dimensional arrays require very different treatment. In a linear, or one dimensional array, faulty elements can simply be bypassed; in a two dimensional array, the network connectivity must be maintained in the presence of the faults and this is a more complex task. In this paper we consider only two dimensional arrays. Secondly, in WSI we are dealing with the unknown in the sense that we do not know which parts of the circuit are functional and which are faulty. It may be that the switches themselves are faulty. We therefore need to investigate ways in which faults can be detected and tolerated, not just in the array itself, but in the configuration logic, the self-test circuitry, and any other circuitry which might be used. In short, we need to develop a verifiably functional system so that the user can be confident that his array will configure itself correctly.

In this paper, we present a hierarchical approach to testing and fault tolerance within the array. This allows the user to verify that the circuit

Copyright © Controller HMSO, London, 1986.

is working and also maximises the array yield. In the following section we briefly review an approach to 2-dimensional WSI processor arrays in which the functional elements have the ability to organise themselves around the faulty ones in order to construct a functional array. This is followed by a discussion of the requirements and potential problems of testing the array elements and the control circuitry and a presentation of a hierarchical strategy which permits external testing of all the control circuitry with the emphasis on verifiability by the user. We also show how faults in the control circuitry and test circuitry can be tolerated. In the final section we attempt to quantify the effect of the hierarchical test strategy on the overall array yield.

THE 'WINNER' SELF-ORGANISING ALGORITHM

'WINNER' is an acronym for 'Wafer Integration by Nearest Neighbour Electrical Reconfiguration', and is an algorithm for configuring a 2-dimensional array of processors in the presence of faults; see Evans et al (1985). The central concept of the algorithm involves distributing a small amount of control circuitry throughout a 2-dimensional array of processing elements such that each processor has an identical extra circuit associated with it. The extra

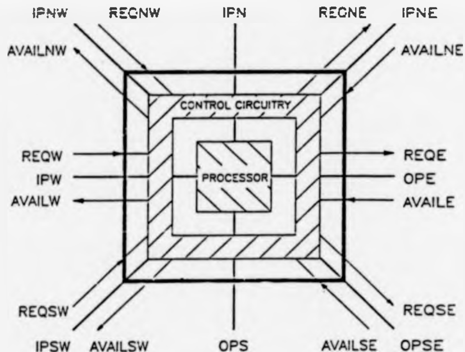


Figure 1. Connectivity of Self-Organising Cell.

circuitry gives the processors the ability to decide how they should be connected to their neighbouring processors based on a knowledge of their neighbours' functionality, and availability. The interaction between processors occurs locally, with processors communicating only with nearest neighbours and this results in a complete self-organisation of the array into a functional array.

A cell with interconnections suitable for generating orthogonally interconnected arrays of processors is illustrated in figure 1. It can be seen that in addition to the North, South, East and West connections normally required for an orthogonally interconnected array, the cell has North-West, North-East, South-West and South-East connections. These increase the connectivity of the cell and allow faulty cells to be avoided. Furthermore, there are connections prefixed by REQ and AVAIL. These are single bit signals which allow the control circuits in adjacent cells to interact with each other. The manner in which these interactions take place forms the heart of the WINNER self-organising algorithm, and is presented in table 1 in the form of a truth table giving the cell AVAILability outputs corresponding to REQuest inputs, and vice-versa.

Table 1. Generation of Availability and Request Signals

Cell must be functional, and at least one REQuest or AVAILability input must be TRUE, otherwise REQuest and AVAILability outputs become FALSE. (X = DON'T CARE)

REQUEST INPUTS			AVAILABILITY OUTPUTS		
REQNW	REQW	REQSW	AVAILNW	AVAILW	AVAILSW
TRUE	X	X	TRUE	FALSE	FALSE
FALSE	TRUE	X	TRUE	TRUE	FALSE
FALSE	FALSE	TRUE	TRUE	TRUE	TRUE
FALSE	FALSE	FALSE	TRUE	TRUE	TRUE

AVAILABILITY INPUTS			REQUEST OUTPUTS		
AVAILNE	AVAILE	AVAILSE	REQNE	REQE	REQSE
TRUE	X	X	TRUE	FALSE	FALSE
FALSE	TRUE	X	FALSE	TRUE	FALSE
FALSE	FALSE	TRUE	FALSE	FALSE	TRUE
FALSE	FALSE	FALSE	FALSE	FALSE	FALSE

FOR BOUNDARY INPUTS: HORIZONTAL INPUTS = TRUE
DIAGONAL INPUTS = FALSE

The local decisions made by the cells within the array occur simultaneously. In the early stages of self-organisation the situation may be highly dynamic with cells forming and relinquishing connections to other cells as a result of being overridden by higher priority decisions which have been made at other localities and have propagated through the array. However, although connections may experience a number of iterations of this type, stable functional rows of interconnected processors are formed, one by one, starting at the top of the array. Once formed, these rows are no longer affected by the activity of the cells in the lower parts of the array. In effect, a 'boundary' moves through the array from top to bottom, above which stable rows

exist, and below which stable rows have yet to be formed. When the boundary passes out of the bottom of the array, all possible functional rows will have been formed. This process is always completed within a fixed number of steps - approximately 2N, where N is the dimension of the array. The priorities for REQuest and AVAILability which are given in table 1 ensure that no unresolvable contention problems can occur.

The interactions of the REQuest and AVAILability signals described above generate functional rows of interconnected processors spanning the entire width of the array. These rows have one processor in each column of the array. To form a functional array the rows can be connected together in the vertical direction by making connections between all the cells in a given column, but bypassing faulty cells and cells which, although functional, are not part of a functional row. An array which has been configured in this way is illustrated in figure 2.

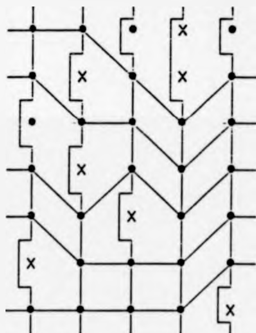


Figure 2. Example of a Configured Array.

Simulations to estimate the performance of the algorithm have been carried out and the results are plotted in figure 3. The graph shows a number of curves, each representing a constant value of array yield. These show how the array yield varies with both redundancy overhead and processor yield and indicates that processor yields of 60% or greater are likely to be required in a practical system.

HIERARCHICAL TEST STRATEGY

From the manufacturers point of view, the ideal WSI system would be one in which the array elements are capable of performing a full self-test, and configuring themselves automatically into a functional array with 100% reliability and without any external assistance. In the real world such a

system can only be a dream since we cannot rely on any part of the circuitry on the wafer to perform its predefined task correctly. From a practical point of view this means that the manufacturer must perform at least a small test on some part of the circuitry. The tested circuitry can then be relied upon and used in further tests of the wafer. The challenge is to develop a strategy which requires a small amount of circuitry to be externally tested, and to be able to carry out the test in a simple manner.

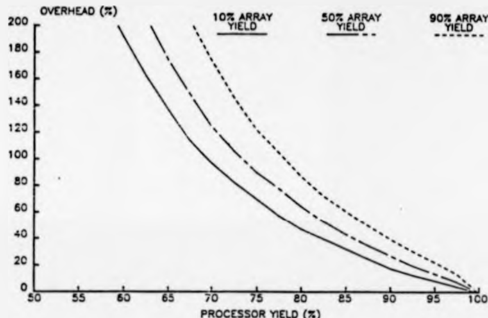


Figure 3. Performance of the WINNER Algorithm.

Although essential to the self-organising algorithm, the control circuitry can cause serious degradation in the overall array yield for the following reason. The self-organising array consists of a number of processing elements together with their associated control circuits. Provided that sufficient spare rows have been used, the configuration algorithm can potentially configure a functional array even if many of the processors are faulty. However, for an array to be sure of working, all the control circuits must work, and although each is very simple, the total amount of control circuitry in the array is quite large. It is easy to see that in a large array, the array yield achieved is likely to be dominated by the possibly poor yield of the array of control circuits rather than the configured array of processors. The test and fault tolerant hierarchy to be described reduces this problem to a more acceptable level.

The testing strategy operates at several levels. The highest level is the self-test performed automatically by the elements of the array. This is assumed to be based on the signature analysis approach. The other levels of test are all performed by an external source and allow testing of the control circuitry, the signature analyser comparator and the scan paths which are used to test the control circuits. In addition, faults in the control circuits and comparator can be tolerated by masking out the circuits which are faulty so that they are ignored by the other elements within the array. Faults in the scan paths can also be tolerated to some extent as will be described.

Control Circuit Tests

In considering the self-organising algorithm described in section 2, it can be seen that a fault in a control circuit could be disastrous. For example, an AVAILability output in a cell containing a faulty processor could be stuck-at-1, incorrectly indicating that the cell is available for use. This could cause the faulty processor to be inadvertently configured into the array and would result in the array being non-functional. For this reason, an external test of the control circuitry is essential.

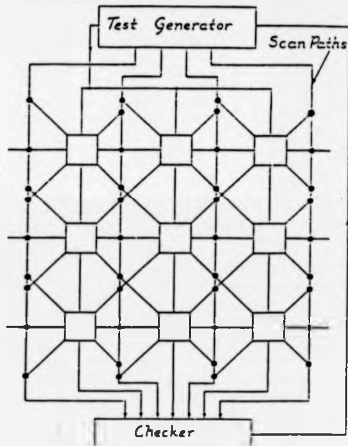


Figure 4. Schematic of the Scan Path Testing Approach.

The control circuitry associated with each element of the array is a purely combinational circuit and as such can be tested easily if it can be accessed from an external source. The required access can be provided by including a scan path between each column of processors as illustrated in figure 4, in which each dot represents a group of scan path registers in the AVAILability, REQuest and signal paths. A single scan register is illustrated in figure 5 and its function for different values of A and B is shown in table 2 (the AND gate should be ignored for the moment). By applying the appropriate combination of control signals at the A, B and clock inputs, test patterns can be loaded serially into the scan paths from the external tester, (set A=0, B=0), applied in parallel to all the control circuits (A=1, B=0), and the outputs clocked out serially for checking (A=0, B=0). This procedure allows all control circuit faults to be detected with a small number of test

patterns. To allow the control circuitry to operate normally, A and B are both set to 1 to allow signals to pass straight through the scan register.

Table 2. Scan Path Function.

A	B	Scan Function
0	0	Serial-load shift register
0	1	Serial test of straight through connection
1	0	Parallel-load shift register
1	1	Activate straight through path

Having detected a faulty control circuit it is desirable to be able to tolerate it rather than discard the whole wafer because of a small fault. From table 1, it can be seen that a cell receiving a FALSE AVAILability signal from its neighbour cannot output a REQuest to that neighbour. Furthermore, a cell receiving a FALSE REQuest input from a neighbour is not inflenced at all by that neighbour. As a result, if REQuest and AVAILability outputs of a cell containing a faulty control circuit could be forced to be FALSE, it would effectively mask the faulty cell out of the array. This can be achieved by using a modified scan path register in which the link between the upper and lower multiplexers is replaced by an AND gate as shown dotted in figure 6. During the test phase, the register operates in exactly the same way as an ordinary scan path register, but in the operational mode (A=1, B=1), the output signal can be controlled either to follow the input signal or to output a permanent FALSE value; this is achieved by preloading the register from the external tester with a 1 or 0 level repectively.

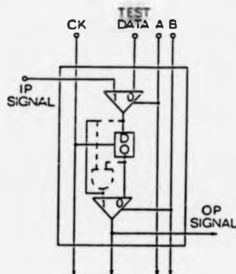


Figure 5. Modified Scan Register for Tolerating Control Circuit Faults.

The components within the scan path registers themselves can be thoroughly tested by the external tester by simple test patterns. This ensures that they can be relied upon to force REQuest and AVAIlability signals to FALSE when controlled to do so.

Signature Comparator Test

A potential problem with processors which have on-board self-test is that even if the self-test result shows the processor to be functional, there is still the possibility that the signature comparator is at fault. This problem can be overcome by testing the comparator from an external source using the scan paths registers descibed above. We assume that the signature of the self-test is formed in a register of some kind and can be clocked serially into the comparator. To test the comparator, we need to temporarily break the serial connection to allow test patterns to be injected into the comparator. The test pattern required will depend upon the structure of the comparator, but in principle it is possible to carry out a full test which checks that a pass indication is only delivered by the comparator when the correct signature is applied. The results of the test can be monitored via the scan paths. The comparator test could be carried out at the same time as the test on the control circuitry, and as with the control circuit test, if a comparator is found to be faulty, the outputs of the cell containing the comparator are forced to zero before configuration takes place.

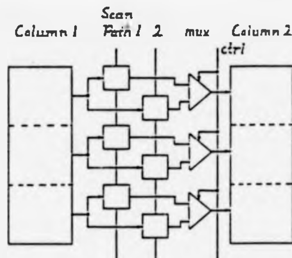


Figure 6. Fault Tolerant Scan Paths.

So far we have shown how to fully test the scan paths, the control circuits and the comparator. However, the amount of circuitry required in the scan paths is not insignificant and their yield may therefore be less than desired. This can be improved by noting that the scan paths can be duplicated to introduce fault tolerance into the paths. Instead of placing a single scan path between each column of cells in the array, two scan paths are used, as shown in figure 6. After the two scan paths have been externally tested, one of the two scan paths can be selected for use by appropriately controlling a column of multiplexers associated with each pair of scan paths.

EFFECT ON ARRAY YIELD

In order to assess the benefits of introducing the proposed testing strategy into the array we need to consider its effect on the overall yield of the array. This is not an easy task because of the difficulty of making realistic estimates of yields of individual components. However in order to make a resonable comparison we have made the following assumptions:

Processor Complexity	10,000 gates
Processor Yield	65%
Target Array Size	10 by 10
Redundancy Overhead	100%
Control Circuit Complexity	30 gates/cell
Comparator Complexity	50 gates/cell
Scan Path Complexity	840 gates/column

The probability that a single gate works is estimated as follows. We assume that each processor has an independent probability of working of 0.65, and so the probability that a single gate works is therefore $Y_G = 0.65^{1/10,000}$. This figure can now be used when estimating the yields of control circuits, etc.

In a 10 by 10 target array operating with 100% redundant elements, there are 200 cells. In the absence of any technique to tolerate faults in the control circuit or comparator, the yield of the control/comparator array would be $Y_G^{200.80} - 0.5$. This means that however good the configuration algorithm might be at generating functional arrays, the array yield cannot be higher than 50%.

The inclusion of the scan paths in the array alters this situation, since now the control/comparator circuits do not all have to work. The yield of the control circuits is now reflected in the processor yield which is slightly reduced from 0.65 to about 0.648. However, all the scan paths must work if the array is to have any chance of configuring correctly. The total scan path circuitry in an array with 200 cells is about 8400 gates. The probability that the whole array of scan paths is functional is therefore 0.7, which is better than the figure of 0.5 for the control circuitry alone but not really acceptable. A significant improvement in scan path yield can be achieved however by placing two scan paths instead of one in between each column of cells. The probability that a single column of scan path registers works is about 0.965 and that of one column scan path out of two being functional is 0.998. The figure for each column of multiplexers required to select the functional scan path is about 0.993. When all the columns are considered, the probability of the array of scan paths working becomes 0.92, which is a great improvement on the figure of 0.5 for the control circuitry alone.

CONCLUSION:

We have described a hierarchical approach to testing a wafer scale, two-dimensional array which is to be configured using the WINNER algorithm. The technique also includes the ability to tolerate faults in a very simple manner at a number of levels within the circuit including the processors, the

self-test signature comparator, the control circuitry, and scan path registers. We have shown that the use of the combination of fault tolerant techniques achieves a much improved probability of an array being functional. In addition, the user can be much more confident that his system is fully functional than he could before, because he can now verify by simple tests that each section of the configuration and test logic is functional.

REFERENCES

Chevalier G and Saucier G, 'A Programmable Switch for Fault Tolerant WSI of Processor Arrays' Proc WSI Workshop, Southampton, UK, July 1985.

Evans R A, McCanny J V and Wood K, 'Wafer Scale Integration Based on Self-Organisation', ibid.

Moore W R and Mahat R, 'Fault Tolerant Communications for WSI of a Processor Array' Microelectronics and Reliability, 25, 2, pp291-294.

Raffel J I, 'The RVLSI Approach to WSI', Proc WSI Workshop, Southampton, UK, July 1985.

THE BRITISH LIBRARY DOCUMENT SUPPLY CENTRE

TITLE

Self-organising Techniques
for Tolerating Faults in
2-Dimensional Processor Arrays

AUTHOR

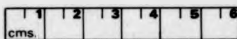
Richard Anthony Evans

INSTITUTION
and DATE

University of Warwick
1988

Attention is drawn to the fact that the copyright of this thesis rests with its author.

This copy of the thesis has been supplied on condition that anyone who consults it is understood to recognise that its copyright rests with its author and that no information derived from it may be published without the author's prior written consent.



THE BRITISH LIBRARY
DOCUMENT SUPPLY CENTRE
Boston Spa, Wetherby
West Yorkshire
United Kingdom

REDUCTION X 20

CAMERA 8