A RECONFIGURABLE ARCHITECTURE FOR VERY LARGE SCALE MICROELECTRONIC SYSTEMS

Wei Chen

Doctor of Philosophy

Department of Electrical Engineering

University of Edinburgh

December 1986

ABSTRACT

With the inexorable increase in integrated circuit complexity and chip size, an efficient defect tolerant method must be incorporated into the design of large silicon integrated systems to avoid dramatic yield loss. Furthermore, it is highly desirable that future integrated systems be high performance, low cost, and particularly have system level reconfigurability. This thesis introduces an architecture embodied in a large silicon chip, called *superchip*, which can be tailored by the user to a specific system to perform a particular processing task. The methodology of system design using the superchip architecture is presented both at the top level of system organisation and at a lower level of system customisation, through a suite of supporting software. The superchip architecture offers defect/fault tolerant capability and system reconfigurability by incorporating a crossbar switching network in the system. This crossbar switching network connects all the processing elements in the superchip together to accommodate the required communication. Defect/fault tolerance is achieved by introducing redundancy through the switching network.

Cost-effectiveness is one of the major issues investigated in the thesis. By employing the optimal redundancy predicted by yield models and the redundancy selector developed during this research, a dramatic yield improvement over the yield without redundancy can be achieved. This brings the superchip yield up to an economically acceptable level, while keeping the hardware overhead at a minimum.

Finally, an example is given to illustrate the design and customisation process for implementing an FFT system in the superchip style. High performance and flexibility are achieved by reconfigurability of the architecture. The example system, having $35 \times 35\ mm^2$ silicon area, is capable of approximately 150 million arithmetic operations per second. More powerful systems can be achieved by extending the concept to an entire silicon wafer by using the same processing technology, or by increasing the density using a smaller geometry processing technology.

Declaration of Originality

The material contained in this thesis was researched and composed entirely by myself in the Department of Electrical Engineering, University of Edinburgh between March 1984 and December 1986.

Signed

Wei Chen

ACKNOWLEDGMENTS

I would like to thank my supervisors Professor John Mavor and Dr. David Renshaw, and also Professor Peter Denyer for their continuous encouragement and technical guidance throughout the research.

Thanks also go to the members of the Integrated Systems Group, particularly Dr. Kunihiro Asada, David Fletcher, Stewart Smith, and C. H. Lau, for their assistance and conversation. Dr. Gordon Brebner also has my thanks for his interest and contribution on the mathematical proof of NP-completeness for the routing problem. I am grateful to SERC for their silicon brokerage service. Thanks also go to University of Edinburgh and Shanghai Jiao-tong University for their financial support, and CVCP for the ORS award.

Finally, it is a pleasure to acknowledge my parents' support during my stay in Edinburgh.

CONTENTS

1 INTRODUCTION 1

Introduction 1

A Brief Review of Defect-Tolerant Techniques in Integrated Systems .. 4

The Defect/Fault Tolerant Approach Introduced in This Research 10

Outline of Thesis 10

2 A RECONFIGURATION AND DEFECT/FAULT TOLERANT MONOTHIC COMPUTING SYSTEM 13

Introduction 13

Crossbar vs Other Interconnection Networks 13

The FIRST Silicon Compiler 19

The Superchip Architecture 20

Summary 35

3 YIELD MODELLING AND REDUNDANCY SELECTION 36

Introduction 36

Previous Work on Yield Modelling 36

Superchip Yield Modelling 47

Optimal System Block Partitioning Using Crossbar Networks 71

Block Partitioning versus Hierarchical Redundancy 74

Conclusions 77

4 THE PROTOTYPE CROSSBAR SWITCH MATRIX DESIGN AND EXPERIMENTAL RESULTS 79

Introduction 79

Prototype Design Procedure 79

Test Procedure of the Prototypes 83

Test Results 87

Design Deficiencies 107

Summary 112

5 A MESSAGE ROUTING ALGORITHM FOR THE CROSSBAR SWITCH MATRIX IN THE PRESENCE OF FAULTY SWITCHES 113
Introduction 113
The Problem and Solution Tactics 113
Goodness Function for Net Assignment 117
The Algorithm 123
Implementation 130
An Example 133
Discussion 143
Conclusion 143

6 A COLUMN FFT ENGINE USING THE CROSSBAR SWITCH MATRIX – A CASE STUDY 145
Fast Fourier Transform 145
A Monolithic 64 Point Column FFT System 147
Expandability of the FFT Superchip 158
Conclusion 168

7 CONCLUSION AND RECOMMENDATIONS FOR FURTHER RESEARCH 169
Summary 169
Recommended Further Research 171

REFERENCES 176

APPENDIX A: SOME MATHEMATICAL NOTATION 190

APPENDIX B: NP-COMPLETENESS OF THE ROUTING PROBLEM ... 192

APPENDIX C: PUBLICATIONS 197

NOTATION

ALU	arithmetic–logic unit
BPE	butterfly processing element
CAD	computer-aided design
CMOS	complementary metal–oxide–semiconductor
CPU	central processing unit
CtrlIn	control input of a processing element
CtrlOut	control output of a processing element
CVD	chemical vapour deposition
DAS	digital analysis system
DFT	discrete fourier transform
DIF	decimation-in-frequency
DSP	digital signal processor
ECL	emitter–coupled logic
EPROM	erasable programmable read–only memory
FET	field–effect transistor
FFT	fast fourier transform
GaAs	gallium arsenide
IC	integrated circuit
LSB	least significant bit
LSI	large scale integration
MSB	most significant bit
NMOS	N channel metal–oxide–semiconductor
PCB	printed–circuit board
PE	processing element
ROM	read–only memory
SA1	stuck at 1
SA0	stuck at 0
SEM	scanning electron microscope
SigIn	signal input of a processing element
SigOut	signal output of a processing element
S_ON	stuck on
S_OP	stuck open
SSI	small scale integration
ULSI	ultra-large scale integration
VLSI	very large scale integration
WSI	wafer-scale integration

CHAPTER 1

INTRODUCTION

1.1. Introduction

The evolution of modern electronic systems from the embryonic stage of vacuum tube realisations at the beginning of this century through discrete transistor, SSI, and LSI realisations to today's VLSI complexity illustrates the trend in system integration towards more compact, cheaper, faster, and more reliable systems. For example, modern VLSI technology has made it possible to integrate hundreds of thousands of transistors on to one small piece of silicon, typically less than 100 mm^2. The integration evolution has matured over the past decade and resulted in the appearance of single chip DSP [1,2] and microprocessors [3,4], very large memory chips [5,6], and so on. Continually, integrated circuit designers have endeavoured to increase chip complexity by integrating more and more circuits on to a single chip with various design styles. The trend nowadays is towards a system-on-a-chip which requires an entire silicon wafer to accommodate the entire electronic system [7,8,9,10]. There are many advantages of having fewer, but larger, IC chips compared to having many smaller ones wired together on a circuit board to form the same system. These include:

- potentially higher speed due to less inter-chip connections. This follows from the fact that on-chip capacitance is much smaller than off-chip capacitance and, therefore, on-chip delay is less than off-chip delay. For current ECL gate arrays, for example, the internal cells are about 2.5 times faster than the inter-chip delays (including the wiring delay). For CMOS this ratio can be as high as 10 times when the internal cells are close to each other† [11,10].

† It might be argued that one can not simply scale this to full wafer products. But generally, as chip complexity increases, this observation still stands.

- lower packaging and assembly cost resulting from the simplification in conventional packaging with fewer bonding pads, gold wires, and package conductor runs, etc. As a result, more silicon is available for computational elements rather than being used for the large drivers required for driving large off-chip capacitive loads [12,13,10]. The pin count per logic function is also reduced for the large single chip system.

- lower power dissipation achieved by having fewer bonding pads. Much lower output capacitances which results lead to the requirement for smaller output drivers [12,10].

- higher reliability expected through having fewer mechanical connections, than is the case when mounting chips on a printed-circuit board [10].

However, problems have emerged in experimental work directed to developing large, single chip, integrated circuits. One of the most important and dominant problems is the rapid degradation in yield which results when the chip size is appreciably increased duction sizes, because, with the increase of integration scale and circuit complexity, catastrophic processing defects on each chip become increasingly likely. These processing defects degrade the overall yield of the chip dramatically and, thus, increase the cost. For example, a well 'tuned' processing line can achieve a defect density of two defects per square centimetre. The probe yield of a typical VLSI chip today with 5×5 mm^2 size can be around 60%. However, if the chip size is increased to 15×15 mm^2, using the Poisson yield model (which is rather pessimistic in practice but often used as an analytical tool for yield prediction), the new yield could be as low as 1%. The IC industry can not afford such implied yield losses in the exploration for higher levels of integration. This is a prime consideration in setting the optimum size of chip for a particular IC process. Naturally, yield enhancement can be expected by continual improvements in IC processing which lower defect density, but such improvements are gradual and very expensive to achieve. However, today's VLSI technology has made it possible to afford a degree of redundancy, which enables defect tolerance to be included in chips to achieve acceptable yields, that current technology can not offer. Therefore, it is more feasible to enhance yield by designing defect-tolerant integrated systems, than wait for improvements in the technology. So far, the accepted approach to larger chips is to introduce some form of hardware redundancy into the circuits. Thus, affected elements can be switched out via some programmable switches and replaced by fully working

circuit sections.

Besides low yield, there have been some other problems in large integrated systems; one being reliability. With the increase of system complexity, the probability of having an operational fault (often classified as soft error or transient error) at any time also increases. It has been estimated [14] that a system containing two thousand VLSI chips would suffer from transient failures at a rate of one every 50 hours and permanent hardware failures at a rate of one every 500 hours of operation. For larger silicon chips, occasional operational faults of some elements in the chip are still likely during field operation, although their reliability is better than the same system with smaller chips because of the level of integration. Such occasional faults in some elements may cause a great deal of performance degradation as well as malfunctioning of the whole system. To improve reliability for complex electronic systems, the conventional fault-tolerant approaches, such as TMR (Triple Modular Redundancy) [15,16], are required. However, due to the large hardware overhead, such a technique will become more and more costly in future.

The approach advocated in this thesis is to achieve fault tolerance for a chip by avoiding the faulty elements whenever they appear, rather than discarding the whole system. To be consistent in terminology throughout the thesis, methods† of circumventing defects in the chip are referred to as *defect tolerance*. The method of recovering the system from an operational fault is called *fault tolerance*. The replacement of a faulty element during system operation can be achieved by incorporating hardware redundancy, by introducing a switching network into the system.

Another problem, or perhaps desire, has been the issue of system reconfigurability. There has been a great deal of interest in the design of reconfigurable architectures whose paths between elements (processors, memory and I/O devices) can be reconfigured by some programmable control. Such reconfiguration can be achieved in two ways according to different tasks and interconnection networks. One is to set up interconnection paths once, and prior to the initiation of the tasks, or at compile time††; this is here called *static reconfiguration*. The other is to reconfigure the interconnection paths

† These expressions are in common use [17].

†† This is derived from an analogy with conventional programming. High-level language programs are compiled to produce executable object code. Similarly, some specification of the communication requirements of an algorithm can be compiled to produce control signals to set up the network prior to execution [18].

'on-the-fly', or at run time; this is here called *dynamic reconfiguration*. The latter is particularly desirable in applications where the communication patterns have to be changed during the operation, or they are data-dependent. To achieve such flexible reconfiguration schemes, a good interconnection network with simple and easy control is desired.

This thesis addresses the key three issues of yield, reliability and system reconfigurability, by incorporating a flexible crossbar switching network into integrated systems. This course should improve yield for very large silicon chips, and achieve system reconfiguration, either statically or dynamically. A general system architecture is proposed which consists of multiple processing elements (PEs) and a crossbar switching network. It is capable of performing a range of tasks by programming the crossbar switching network to customise the system. This programming procedure can be performed either before the system is operational; static reconfiguration, or during operations as well; dynamic reconfiguration. Static reconfiguration is often preferred when only system customisation and/or defect-tolerance are to be achieved. Such initial system interconnection is kept unchanged during the operation. Defect-tolerance is achieved by connecting all the necessary, and working processing elements together using working switches in the crossbar switching network. Dynamic reconfiguration, on the other hand, is usually performed when the connections in the system need to be changed or a fault is found during its operation. The objective of this thesis is to assess the potential of employing a crossbar switching network in very large integrated systems by considering the following factors: interconnection organisation, defect-tolerance and fault-tolerance.

1.2. A Brief Review of Defect-Tolerant Techniques in Integrated Systems

Processing defects exhibit themselves in many different ways resulting in a variety of fault conditions. These processing defects can be classified in two categories. One is referred to as a point defect. Each such defect is a local phenomenon affecting only the microstructure in its own immediate area. Pinholes in the oxide layer belong to the point defect category. Processing defects in the other category have a size comparable with the geometry of circuit layout. They are usually caused by contamination introduced during the processing, typically, resulting in open- or short-circuit conditions. Typical defects have been analysed and listed by a variety of authors [19,11,20,21,22]. Table 1.1 lists some of the defect types for current double-metal VLSI technology. For

each type of defect the likely mechanism causing the defect is also listed.

Table 1.1 Classification of Defect Types

Fault Condition	Equivalent Source Defects
unexpected opens or shorts among features on two metal layers	large contamination spot during the process
unexpected opens or shorts among features on polysilicon layer	contamination during processing
unexpected opens or shorts among features on diffusion layer	contamination and schedule errors in implantation
poor contact and via	high contact resistance or poor metal coverage over contact regions
unexpected shorts between features on metal and polysilicon layers	poor oxide quality pinholes in the oxide
unexpected shorts between features on metal and diffusion layers	poor oxide quality pinholes in the oxide

Any of these defects identified in Table 1.1 may cause a logical malfunctioning of the circuit. However, introducing hardware redundancy in design to replace the defective logic area is the common approach to circumvent such defects. In practice, the part of the circuit affected may be locally circumvented by disconnecting it from the whole circuit and connecting (switching in) a working redundant logic part into the circuit. This idea was originally introduced in the design of large memory chips, because they are highly regular in design and densely packed and, therefore, are highly sensitive to defects. However, their highly regular architecture makes it possible to efficiently implement various redundant elements as a standby to replace any affected element. Various large memory designs with different defect tolerant schemes have

been presented [23,24,25,26]. The techniques for implementing defect-tolerant schemes vary with the different switch implementations for introducing redundancy. The switches can be classified as *physically programmable* switches or *electrically programmable* switches. The physically programmable switches are realised by incorporating special links on the chip between sections of circuit. The switch setting is performed using non-electrical methods, such as a laser, to make the special links conducting or non-conducting. This approach usually needs special processing steps or programming equipment. The electrically programmable switches are usually realised using transistor structures within the chip. For this approach the switch setting is performed at circuit level by turning on or off transistors using electrical control. Some typical defect-tolerant techniques using either physically or electrically programmable switches are summarised in the remainder of this sub-section.

1.2.1 Discretionary Wiring

A large integrated system can be divided into many functional units. These functional units are distributed and processed discretely over a large silicon area like many chips on a wafer. After processing, each functional unit is tested. The discretionary wiring technique is used to generate correct interconnection patterns by linking the working functional units on the silicon to form a desired large integrated system. Defect-tolerance is achieved by isolating the defective functional units from the system. When the wiring pattern is complicated, the resulting systems are small, reliable and potentially cheap comparing to the printed-circuit board approach using individual packaged chips. However discretionary wiring requires high quality processing steps to generate metal wiring patterns on the silicon. If these wires are regarded as a set of switch links, then this technique provides non-volatile† and non-reversible†† switching.

One of the earliest attempts to use the discretionary wiring technique to build a large memory chip was in the 1960s, when Texas Instruments Inc. produced a prototype 32K memory product on a 2-inch wafer[7]. With this early approach redundant circuits, implemented on a ceramic substrate, were connected into systems by discretionary

† A programmable switch is said to be *volatile* if its control store is volatile, i.e. testing and reconfiguration (switch setting) must be initiated each time power has been removed from the part in question. Otherwise, it is *non-volatile*.
†† A programmable switch is said to be *reversible* if it can be turned off after it has been turned on, or vice versa.

wiring. Unfortunately, this technique was ahead its time and was not then adopted in production. The reason was that complex wiring processes were not mature at the time of its introduction and, therefore, could not be used to produce economically viable large integrated systems. However, this approach has been recently reintroduced [27] by using modern silicon fabrication and CAD techniques, including E-beam writing, thick film/metal structures with relaxed design rules for wiring patterns, and sophisticated CAD support.

1.2.2. Laser Programming

Laser programming techniques include laser welding and laser cutting. Typically, a high energy laser beam is placed precisely on some points or tracks, like bus or power lines, fuse, or antifuse†, etc, to make connections or disconnections. These areas receiving the laser beam usually use special material and include features to help the welding or cutting processes. Defective devices within the system can be disconnected and replaced by a duplicate circuit using this technique. As compared to the discretionary wiring technique, laser programming is easier to achieve and more interactive and flexible to designers, because laser programming facilities are simpler and more accessible to designers. However, it requires some non-standard processing steps.

The Lincoln Laboratory of MIT demonstrated a so-called "restructurable VLSI" architecture [28,29,30] using a laser programmed, vertical metal, H-A-Si (Hydrogenated Amorphous Silicon) sandwich structure to avoid defective elements in the chip. A single wafer, 16-point FFT processor has been designed [31,32] with this approach. Other forms of laser programming include laser-induced CVD conductors [33], and laser programmed polysilicon doping. Laser programming has been used in large commercial memory designs by Bell Laboratories [34,35,36], Hitachi [37], Toshiba [38], and more recently by Mitsubishi [25] and Mostek [39].

---

† Fuse (antifuse) is a linking device. According to its type, it is sensitive to laser, current or voltage with a circuit-opening (circuit-closing) fusible part controlled to disconnect (connect) circuit portions.

1.2.3. E-Beam Programming

Use of an electron-beam programmed transistor switch for performing circuit repairing was suggested by Shaver [40], in which the switch is a floating-gate FET. This type of transistor can be fabricated using conventional processing lines and does not usually require process modification. The programming is performed by applying an electron-beam to the floating gate of the transistor, causing a negative potential to exist on its gate. Such a negative potential can turn on an enhancement-mode p-channel FET, or turn off a depletion-mode n-channel FET. In CMOS, both types of FET can be fabricated, so that both normally "on" or "off" switches are available. A 128K bit NMOS electronically programmable ROM has demonstrated the feasibility of this approach to circuit customisation, as well as to repairing and testing [41,42]. A wafer-scale, pipelined processor system using electron-beam programmable switches has also been reported [43]. As well as applying an electron-beam on the floating-gate of a special designed transistor to reconfigure the circuit, it is also possible to use an electron-beam or ion-beam to impinge on the surface of an IC circuit directly (onto normal transistor structures) to alter some characteristics of the transistors, so that system reconfiguration may be achieved [44]. For example, a strong electron-beam can be directed on to a normal transistor (not a special designed floating-gate transistor) to change its characteristics, so that the transistor is, possibly permanently, turned "on" or "off" to switch in or out circuit portions.

1.2.4. Transistor Switches

The use of normal transistors to perform programmable, reversible, volatile switches has been widely studied. Because a normal transistor performs as a volatile switch, its switch settings should be initialised every time the power is re-applied. Such switch elements can also be used to reconfigure the system so that the communication pattern can be changed according to different requirements.

Various circuit switching schemes for incorporating redundancy, testing, and restructuring around defective circuitry have been proposed. Early exploitation was made in [45] to use transistor switches to form a spiral from a hexagonal array of processing elements. This work has been further developed by Shute et al [46,47], in which a small crossbar interconnection network acts for both defect-avoidance and for system communication link-up. Similarly, fault-tolerant two-dimensional processor arrays

employing transistor switches for defect-tolerance and system configuration [48,49,50,51,52,53] have been reported . Memory designs also use such switching techniques to introduce redundancy, for example, a fast bipolar static RAM by IBM [54].

1.2.5. Other Techniques

WHIP (Wafer Hybrid Interconnection Packaging) technology [55,56,57] developed by Mosaic Inc. used two levels of signal wiring, separated by amorphous silicon, to form a general-purpose cross-point switch. This can be electrically (or by laser) programmed to form any desired set of interconnection networks. Pre-tested functional chips are bonded on to a wafer with the signal wiring pattern. Due to the electrical property of amorphous silicon, these wires can be connected to form a particular system by applying a high voltage at the cross-over of any two wires separated by amorphous silicon. The use of a silicon substrate minimises thermal stress, and allows the use of standard IC technology to fabricate dense interconnect wiring. High yield for this signal wiring pattern has been reported [58]. For a very large and complex system (say, more than a 100 chip count) this could be an economic approach as compared to PCB realisation. A similar view has been taken by researchers at Rensselaer Polytechnic [59].

More recently, another hybrid packaging approach was suggested [60] to circumvent the yield and discretionary wiring problems. This advocates putting pre-tested chips into recesses in pre-etched silicon substrates. Thereafter, a normal two layer metallisation is carried out to customise the whole system.

These hybrid packaging approaches have advantages over PCB or discretionary wiring approaches in terms of the flexibility and yield. They also have the potential of mixing MOS and bipolar chips together or even mixing silicon and GaAs chips together, to form larger systems with the flexibility of hybrid integrated circuits and the reliability of monolithic interconnections.

Amongst other approaches are pad relocation [61], MNOS transistors [62,63], floating-gate electrical erasable, chalcogenide glass and bridging-bond links [64]. Table 1.2 gives a summary of various existing switching techniques for defect tolerance.

1.3. The Defect/Fault Tolerant Approach Introduced in This Research

There is an increasing demand for more processing power for applications including super-computing, vision and speech recognition, bank or office management, etc. More flexible and powerful processing systems with millions or even billions of gates, having better performance and less cost will be vital in future. The trend from VLSI to ULSI (Ultra Large Scale Integration) and WSI (Wafer Scale Integration) is inevitable. Such high integration levels imply hundreds of processing elements, such as floating-point processors, memories, caches, together with the necessary interface circuits, to be integrated on the same silicon substrate. However, efficient communication among these processing elements is crucial to the performance of the whole system. Such communication must have high bandwidth, high reliability and flexible reconfigurability. Defect/fault tolerance is a further factor of equal importance.

In this research, the author believes that the volatility and reversibility of the normal transistor is advantageous to realising defect/fault tolerance and, particularly, system reconfiguration. The switches in other defect-tolerant techniques, such as laser or electron-beam programming, cannot entirely meet the requirements, due to their non-reversibility or their requirement for extra complex programming processes. The architecture advocated in this thesis employs an interconnection network using normal transistors to efficiently achieve the requirements for high performance communication among the processing elements, defect/fault tolerance in the system, and system reconfigurability.

1.4. Outline of Thesis

In Chapter 2, a user-programmable general purpose computational architecture is introduced. This uses a large crossbar switching network for system reconfiguration, system communication, and the defect/fault tolerance medium. The crossbar switching network is detailed in Chapter 3 by developing yield models for various defect situations, such as different defect densities and different levels of redundancy. From this work a redundancy selection scheme is proposed. Results generated by testing the crossbar switching network design are discussed in Chapter 4. In Chapter 5, an automatic message routing algorithm in the presence of faulty switches in the crossbar switching network is presented, along with its implementation in software (C-UNIX). Some trials and their results show the efficiency of the algorithm. In Chapter 6,

potential applications of the architecture are discussed. Finally, Chapter 7 summarizes the thesis and some suggestions for further research.

Table 1.2 Summary of Different Defect-Tolerant Techniques

Types	*V**	*R**	*switched mode*	*comments*
discretionary wiring	no	no	connect	need some extra processing steps
laser programmed link	no	once	connect or disconnect	need special fabrication step
laser induced CVD	no	no	connect or disconnect	need special fabrication step
laser or electrical programmed fuse	no	no	disconnect	laser fuse needs special treatment
laser or electrical programmed antifuse	no	no	connect	laser antifuse needs special treatment
E-beam floating-gate transistor	no	yes	connect and disconnect	need special treatment
MNOS transistor	no	yes	connect and disconnect	need special fabrication
active transistor	yes	yes	connect and disconnect	no special treatment
amorphous silicon	no	no	connect	special material and processing steps
bridging-bond links	no	(once)	connect	no special processing steps involved

* Due to the limited space, V is for *Volatility*, and R is for *Reversibility*.

CHAPTER 2

A RECONFIGURABLE AND DEFECT/FAULT TOLERANT MONOLITHIC COMPUTING SYSTEM

2.1. Introduction

Before discussing the architecture of the monolithic computational system introduced here, some issues relating to the system interconnection on which the proposed architecture is based should be addressed. As discussed in Chapter 1, normal transistor switches are favoured in this research to achieve defect-tolerance, as well as fault-tolerance and system reconfiguration. These factors are thought key to the basic requirements for future large computational systems at VLSI density, and beyond.

2.2. Crossbar vs Other Interconnection Networks

Considerable research has been devoted to evaluating the characteristics of various interconnection networks for more than two decades. Many of them have been reviewed in several survey articles [65,66,67,68,69] to which the reader is referred. Here, emphasis is laid on the crossbar network with some differences from the other interconnection networks being pointed out.

A particular interconnection network can usually be categorised according to four aspects [70]: operation mode, control strategy, switching method, and network topology. By further classifying the network in terms of topology [71], a network can be grouped into two categories: *static* and *dynamic*. In a static topology, links between processing elements are fixed by hardwiring or some similar non-volatile connection so that subsequent reconfiguration is impossible. On the other hand, processing elements in a dynamic topology are linked through a set of active switching elements, whose switching mode can be changed. By setting a particular set of switching elements, reconfiguration can be realised. Multistage [72] and crossbar are the two main types of network in

the dynamic group. A three stage banyan network (one kind of multistage interconnection network), and a traditional two-sided normal crossbar network are shown in Figures 2.1 and 2.2, respectively.

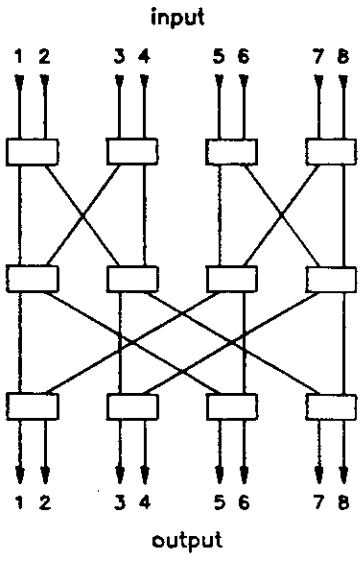

Figure 2.1 An 8×8 Banyan network

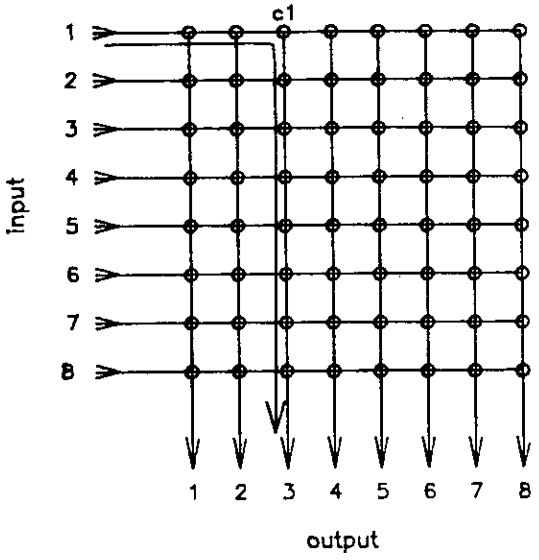

Figure 2.2 An 8×8 Traditional crossbar network

The development of the crossbar switching network was originally motivated by the increasing interest in expanding the availability of the common bus structure. The Burroughs D-825 multiprocessor system was among the earliest systems employing the crossbar interconnection network. One of the major research projects involving a crossbar interconnection network earlier was the "C.mm", the Carnegie-Mellon multi-miniprocessor [73,74,75]. The crossbar interconnection network provides full interconnection capability between any two connected elements, known as ***nonblocking***. It also provides a fast transfer rate and is flexible in term of possible multiple paths between any two connected elements. Real-time data transfer and computation can be carried out within the network. However, early developments in the crossbar approach exhibited a cost growth of $O(N^2)$† for a network having N inputs and N outputs, which is a major obstacle limiting its usage. This is due to the unacceptable implementation cost of N^2 number of switches using MSI (Medium Scale Integration) or LSI (Large Scale Integration). On the other hand, multistage interconnection networks have evolved with slower cost growth over their counterparts in the crossbar network. Various types of multistage interconnection networks have been introduced for connecting N inputs and N outputs. Examples include data manipulator [76], baseline [72,77,78], banyan [79], omega[80], and delta [81], These multistage interconnection networks have a cost growth of $O(NlogN)$, which has been predicted to be the theoretical lower bound for the growth rate of the switch components in the interconnection networks††. However, such an improved cost function is achieved at the expense of increased delay, between inputs and outputs, due to either blocking or the inherent (average) transfer delay††† compared to the crossbar approach. In some applications where speed is not of prime importance, this is an acceptable tradeoff. For situations which require high performance, and 100% realisation of required communications amongst PEs at any time, crossbar networks are more suitable than their multistage counterparts. This is particularly true when the interconnection network is used to construct a system (by customisation) rather than just adding more homogeneous processing elements to achieve parallelism.

---

† Consult Appendix A for the definition of the "big-OH".

†† This argument was given by Shannon [82] in 1950. Interested readers should consult this reference for further details.

††† This is usually caused by the circuit delay of more complex switching elements themselves and more switching elements on a single communication route. However, it may be argued that the communication delay of these two type of networks is of the same order [18].

From a defect-tolerance and fault-tolerance point of view, multistage interconnection networks themselves are much more vulnerable to catastrophic defects in the network. This is mainly attributed to the unique-path property that most of the multistage interconnection networks have. To alleviate this, researchers have modified the networks by adding one or more stages of switches and/or redundant links to the networks [83,84,85], so that multiple paths between inputs and outputs can be achieved. However, such modification can only tolerate defective switches at certain restricted locations (usually the first and the last stages) rather than for any switch in the network. This can not map the real situation where the defects are randomly distributed and all the switches have an equal chance of being defective. Alternatively, the crossbar interconnection network advocated in this thesis is inherently defect-tolerant and fault-tolerant, because of the structure of its multiple-path between any input and output which is able to tolerate defective switches at any location. Due to its regular structure (similar to memory structure), hardware redundancy can be easily and efficiently incorporated into the network. Christos [86] has previously discussed various organisations of the crossbar switching networks to introduce different degrees of fault-tolerance. A conventional two-sided crossbar network is shown in Figure 2.2. The communication path from input 1 to output 3 can be, and only be, made by switching on the switch cell *c1*. Thus any communication path between a given input and output in this configuration is unique, although free from traffic contention†. Such a configuration can not take on a serious defect-tolerance role, because a single failure of any switch cell will rule out one possible input/output communication path. As an alternative, a one-sided duplex crossbar switching network shown in Figure 2.3 introduces the possibility of multiple paths between any input/output pair. With this configuration, all the horizontal conducting tracks serve only internal connections. Inputs and outputs are located either at the top or at the bottom of the network or both. The communication path from input 1 to output 3 can either be made by switching on *c1* and *c2*, or *c3* and *c4*, etc. The number of paths for any input/output pair equals the total number of horizontal conducting tracks.

† In multistage network, implementing several connections may require the same route in a certain stage, blocking implementing these connections at the same time. For example, in the 8×8 banyan network given in Figure 2.1, if we want to implement connections "input 1 → output 1" and "input 3 → output 2", then a traffic contention will happen at the second stage, causing blocking.

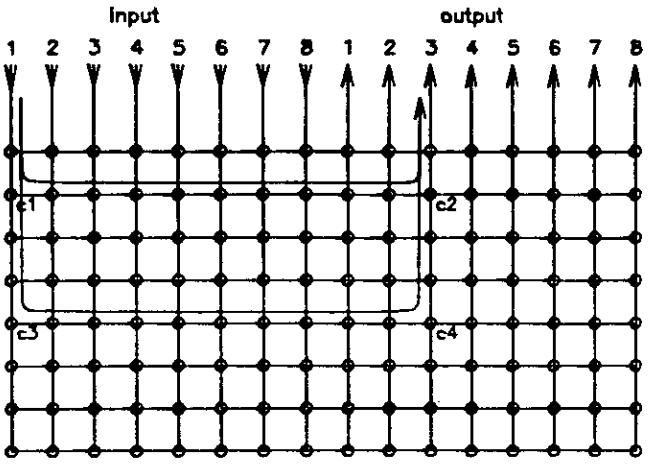

Figure 2.3 An 8×8 Duplex crossbar network

From a system point of view, the switching elements in a multistage interconnection network need to have more switching states (usually more than two) than that for a crossbar network. They need more involved control circuits to address a set of connections than does the crossbar network. In most of the defect-tolerant schemes, critical defects in control circuits are more crucial than those in the switching circuits, because it is difficult to introduce efficiently redundancy or other defect-tolerant schemes into control circuits. Therefore, it follows that simpler control means higher availability of interconnections. It also indicates that the control overhead for systems employing a crossbar interconnection network is less than that for multistage networks. Moreover, the crossbar structure can easily be expanded to a larger interconnection network with smaller networks. This gives system designers more flexibility in adding additional system functions, and/or improving system performance, without redesigning the whole communication network.

The relatively faster cost growth of crossbar networks appears to be a major problem for implementation in comparison to multistage networks. Fortunately, the manufacturing cost of silicon devices (and, therefore, the unit cost of a switch cell in a crossbar network) has gone down rapidly over the last two decades, and is set to reduce with the progress of VLSI technology. As a result, the cost of such interconnection networks will not become the major cost component in the VLSI era, and beyond. As far as the efficiency of integration is concerned, architectural regularity is more important for cost reduction. The previously established cost measure for crossbar networks is no

longer significant in the VLSI era with respect to the cost of multistage networks. Let us consider the banyan switching network as an example of multistage network for comparison with a crossbar network, because many other networks in the multistage network category, such as the Omega network [80], are topologically equivalent to the banyan network [72]. A recent study [87] has already indicated that VLSI realisation of the crossbar network does not necessarily result in a larger silicon area, as compared with the banyan network. They both have a cost growth of $O(N^2)$. The time delay through the network, which is technologically dependent, grows as $O(N)$ for crossbar and $O(N^a(logN)^2)$, where $0<a<1$, for the banyan network. This contrasts with traditional results which predict $O(NlogN)$ and $O(logN)$ for the cost and delay growth of a banyan network [79].

In summary, for future integrated systems which demand higher levels of integration, better performance and lower cost, the crossbar switching network is a suitable candidate for the interconnection network for system communication, defect/fault tolerance and system reconfiguration. The features that the crossbar networks offers can be listed as:

- full interconnection capability (allowing any connection from any input to any output),
- non-blocking,
- constant response time,
- simple control,
- possible system expansion, and
- inherent defect-tolerance and fault-tolerance through redundancy.

The approach advocated in this thesis, towards a defect-tolerant and fault-tolerant architecture for ULSI and WSI is to use a large crossbar switching network as a programmable element to achieve both system communication and defect/fault tolerance. The foundation to this research was inspired by the development of bit-serial architecture employed in the FIRST silicon compiler [88].

2.3. The FIRST Silicon Compiler

In this section, the FIRST silicon compiler and its architecture are briefly reviewed as a useful background to the research undertaken in this thesis.

2.3.1. Bit-Serial Architecture

Bit-serial architectures are distinguished by their communication strategy. Digital signals are transmitted bit sequentially on single wires or busses, as opposed to the simultaneous transmission of the bits of words on parallel busses. This distinction is the key to many inherent advantages of the bit-serial approach as a VLSI strategy. Most significantly, bit-serial transmission leads to efficient communication within and between VLSI chips. This is an outstanding advantage where communication issues dominate, as in many signal processing applications.

As discussed earlier, with the increase in chip size, defect/fault tolerance is a crucial factor in system performance and, particularly, economic production. Owing to the ease of bit-serial communication relative to parallel, the architectural restriction of nearest-neighbour communication need not be imposed, leading to a much greater architectural (and therefore algorithmic) flexibility. Additionally, bit-serial processing elements are generally smaller and therefore intrinsically higher yielding than their bit-parallel counterparts. It has been discovered [89] that fault coverage of bit-serial circuits can be high, and may be determined without extensive fault simulation. The cost in silicon, power, complexity and design difficulty is low, leading to a better yield of function per unit silicon area and high fault coverage for the whole system. Furthermore, introducing pipelined processing in bit-serial systems can result in a better $area*time^2$ measure over its bit-parallel counterparts. Therefore, bit-serial architectures are an attractive proposition, and can offer architectural and defect/fault-tolerant advantages over their bit-parallel counterparts.

2.3.2. The FIRST Silicon Compiler [88]

Silicon compiler is a tool which takes a high level functional description of a system as input but produces, instead of machine code like a conventional compiler, a detailed chip mask geometry. The FIRST (for Fast Implementation of Real-time Signal Transforms) silicon compiler is simply an interpretation at one technology level of the

bit-serial approach. A chip generated by FIRST comprises a central communication channel, flanked by two rows of bit-serial primitive modules. Signal routing is only implemented through the central channel, and there is no intimate connection between neighbouring modules, Thus processing elements communicate by receiving and transmitting data via a routing channel. Chip input and output signals are fed to peripheral pads via the ends of the channel. A channel router is used to carry out the inter-PE routing. Figure 2.4 shows a floorplan of an actual design generated by FIRST, in which the characteristic features are identified.

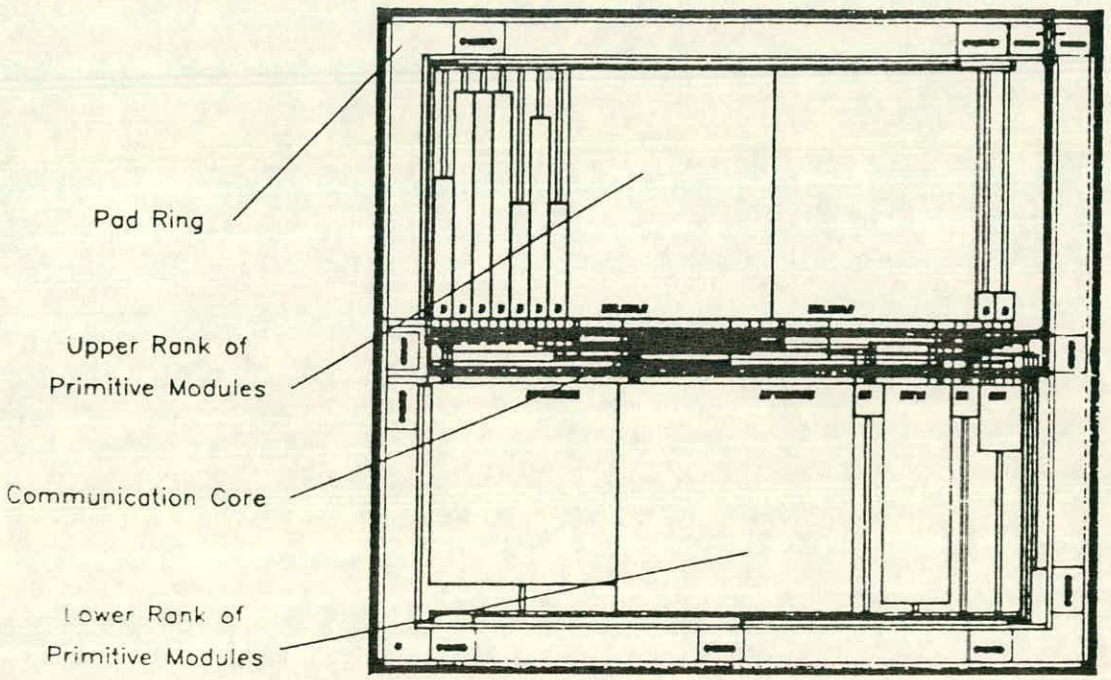

Figure 2.4 The floorplan of a design generated by FIRST

2.4. The Superchip Architecture

A system created by FIRST is not reconfigurable; instead, algorithms are hard-wired through a central routing channel (interconnection network with static topology). The designer can only change the system configuration during compile-time; variations are impossible at run-time. Further, in line with the familiar yield problem, a single catastrophic fabrication defect will result in abandonment of the entire chip. Finally, the requirement for increasingly large, high performance, low cost signal processing

systems implies larger chip sizes. Inevitable fabrication defects limit the practical size of these chips and, thus, the attainable performance level for one substrate.

The approach advocated in this research is to extend the flexibility of the FIRST approach to build arbitrarily complex networks of bit-serial systems. This is to be achieved by integrating a crossbar switching network with a large number of bit-serial PEs. Arbitrary interconnections offered by the crossbar network amongst the bit-serial processing elements contrast with alternative nearest-neighbour-only communication strategies, usually advocated for ULSI and WSI. The chip size required to accommodate the proposed architecture is expected to extend beyond today's usual VLSI chip sizes ($< 100\ mm^2$) and, hopefully, to a whole silicon wafer. For convenience in later discussion, the concept of this scale of integration will be referred to as *superchip*, the proposed architecture implemented with the superchip as *superchip architecture*, and the adopted crossbar switching network as *switch matrix*.

The superchip is a large silicon chip containing many independent bit-serial processing elements. Computational networks are produced by the electrically programmable crossbar switch matrix controlled by the end user. The end user may thus customise superchip to implement a range of signal processing or scientific computational tasks. Figure 2.5 is a schematic diagram illustrating the superchip architecture. Bit-serial processing elements are placed in two rows, above and below a crossbar switch matrix. All the processing elements communicate with each other via the crossbar switch matrix. The crossbar switch matrix permits users to program arbitrary functional networks comprising bit-serial processing elements. Redundant processing elements can be introduced easily by adding extra matrix columns to accommodate these redundancies. These can be switched into the system through the switch matrix. Defect/fault tolerance can also be achieved by programming the switch matrix to avoid faulty elements and employing working redundant elements in the chip, whenever necessary. The addressing data stored in EPROM is generated by a software program, called *router*, which combines the system configuration information and the information about the defective elements in the chip to produce the correct communication paths among the processing elements. A key issue in the superchip architecture is the organisation of the switch matrix and its effect on the performance of the whole system.

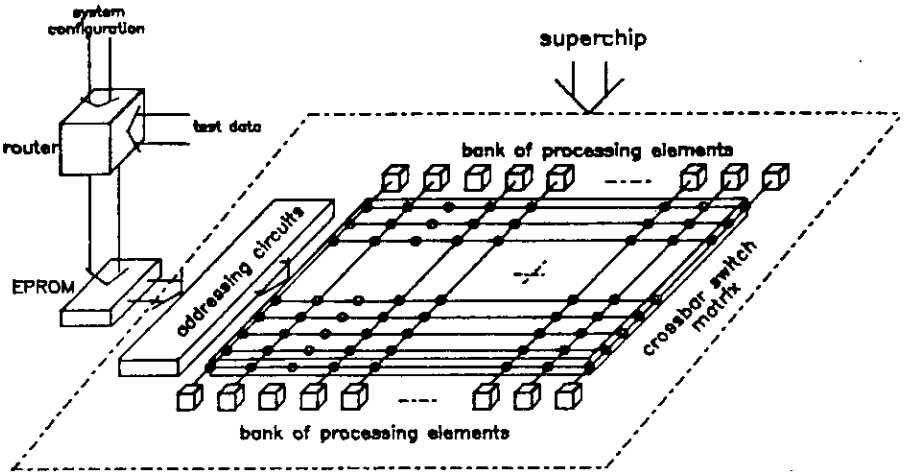

Figure 2.5 A schematic of the superchip architecture

2.4.1. Crossbar Switch Cell and Switch Matrix

Two different switch cells have been considered in this research, and their relative features compared. This involved two different crossbar matrix architectures: two-sided and one-sided architectures, already referred to in Figures 2.1 and 2.3.

In the two-sided switch matrix architecture, one row of the matrix is exclusively assigned to each output of processing elements. Networks are routed in this architecture by programming connections to the inputs of processing elements. Figure 2.6 illustrates the two-sided switch matrix scheme adopted in the superchip architecture, which is similar to the architecture shown in Figure 2.2 except that in this diagram the actual position of the processing elements and the layout style of the matrix are given. The circuit diagram of the switch cell employed is shown in Figure 2.7. This switch cell, let us call it swc-1, comprises a latch, a dynamic NAND gate, and a control pass-gate controlled by the latch. During the precharge cycle, the matrix column (connected to a processing element input) is charged to a logic one. During the evaluation, the matrix column holds its precharged state if the switch is switched off, or becomes the complement of its matrix row (connected to a processing element output) if the switch is switched on. To turn the switch on or off, "column load" is set to logic one so that the switching information from row decoder ("row load" signal) is written into the latch via

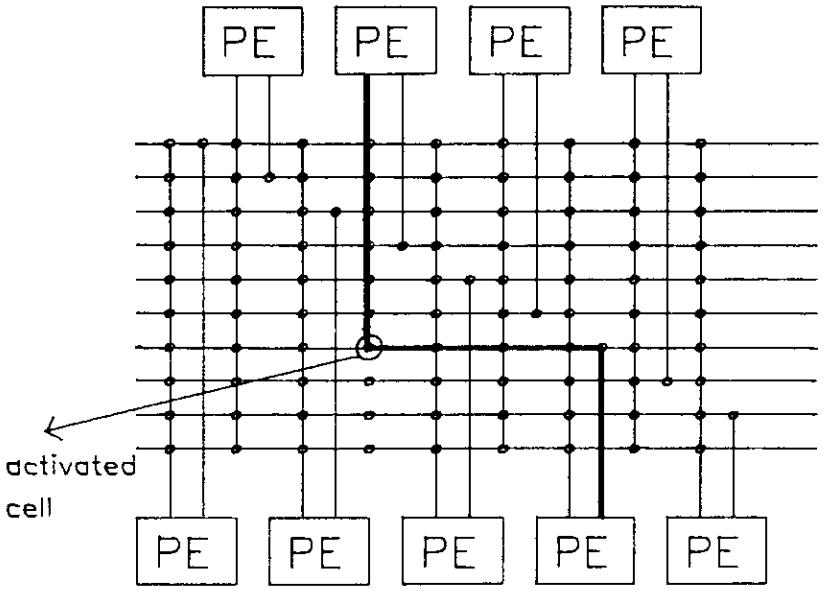

Figure 2.6 Interconnection organisation using traditional crossbar network

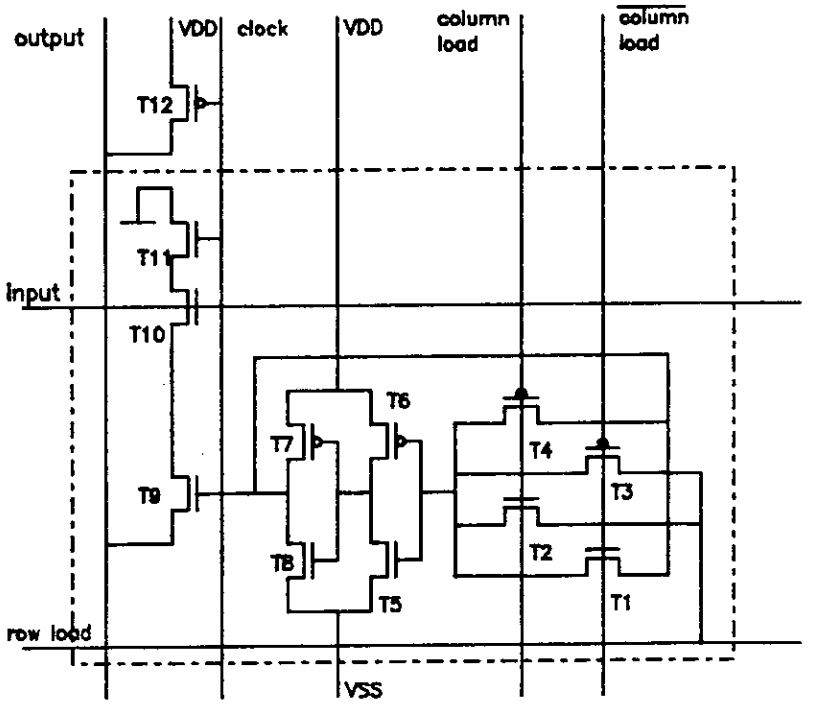

Figure 2.7 Circuit diagram of the dynamic switch cell (swc-1)

the pass gate. Otherwise the latch is isolated from other circuits to maintain the current switching state. At the system level operation of the matrix, the output of each processing element drives one horizontal row of the matrix and every processing element input is driven by a column of the switch matrix. Input and output signals enter and leave the superchip via the switch matrix. Thus input pads are assigned one horizontal channel in the matrix, and output pads each come from one switch matrix column. Clearly, because the switch cell only allows an unidirectional pass of the signal, it is particularly suited to the realisation of the two-sided crossbar architecture. Such a matrix architecture is called *normal matrix* in the rest of the thesis, due to its conventional two-sided matrix organisation.

The alternative switch matrix architecture allows programmable assignment of the horizontal rows to the nets in a system for more flexible routing strategies. Figure 2.8 illustrates this scheme which is similar to the architecture shown in Figure 2.3, except that processing elements are placed at both sides of the matrix. The circuit diagram of this switch cell (called swc-2) is shown in Figure 2.9. Compared to the circuit of swc-1, the dynamic NAND gate is replaced here by a bidirectional transmission gate controlled by the output of the latch. Architecturally, instead of connecting the outputs of processing elements to the matrix rows, each input and output of the processing elements are connected to a column of the switch matrix. Input and output signals of all processing elements enter and leave the superchip via particular columns of the switch matrix. Owing to the use of the bidirectional, static, transmission gate as a switching element signals can pass through the switch cell, from either one direction or the other. Therefore, bidirectional I/O pads are assigned to matrix rows so that each processing element can communicate with external signal sources. Compared to the normal matrix, this matrix duplicates the normal matrix in size. In the rest of this thesis, it will be called the *duplex matrix*.

The performance difference of these two switch cells is determined by the performance of the transmission gate, against that of the dynamic NAND gate. For a quantitative performance comparison, suppose that all the transistors in these two gates have the same parameters, and g_m is the transconductance for a single transistor. For swc-1, the input node of each processing element is discharged through three transistors in series. Therefore the total conductance of the dynamic NAND gate becomes $g_m/3$. Note that precharging is taken to a whole column of the matrix. For swc-2, the total

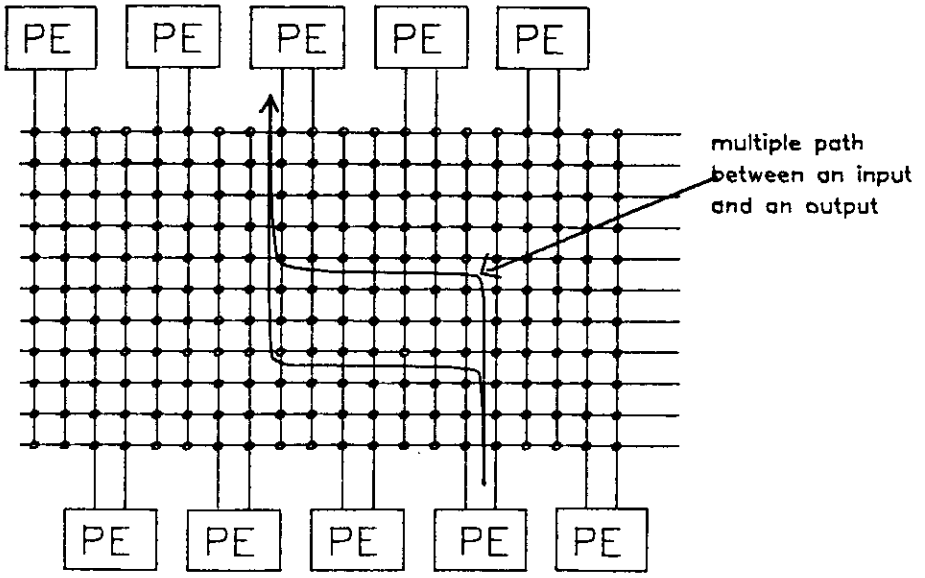

Figure 2.8 Interconnection organisation using duplex crossbar network

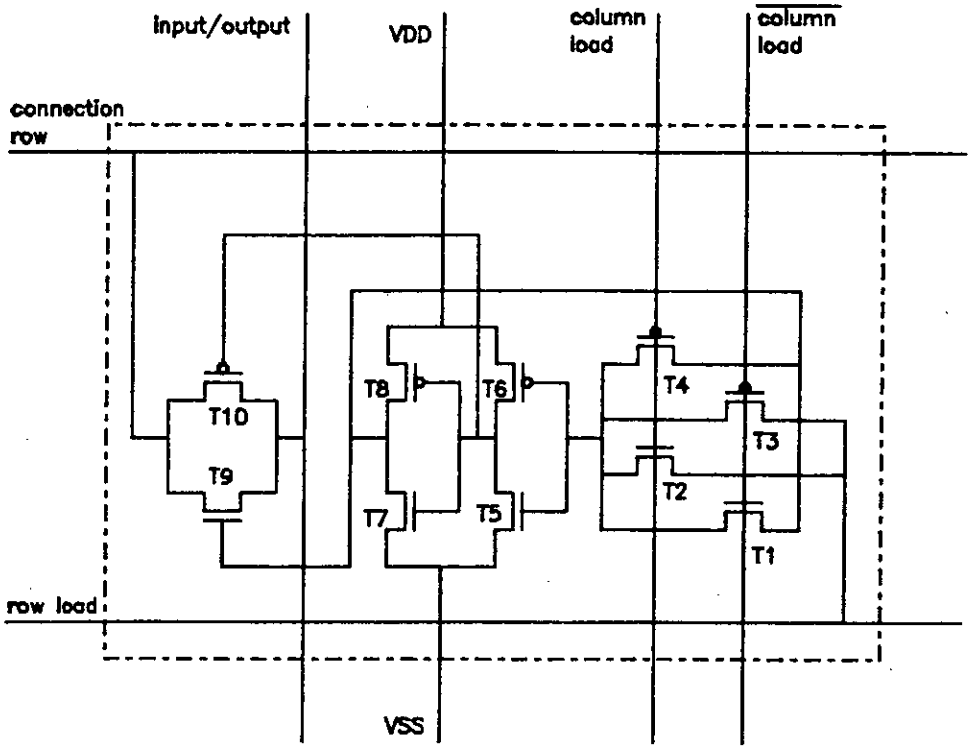

Figure 2.9 Circuit diagram of the static switch cell (swc-2)

conductance of a CMOS transmission gate is equal to $2g_m$ because two transistors are here in parallel. However, because a complete path from an output of a processing element to an input of a processing element passes through two transmission gates in series, then the total conductance of a path is equivalent to the conductance of a single transistor, g_m. On the other hand, the outputs of the processing elements in swc-1 are not connected directly to the inputs of processing elements. A processing element needs only to drive a long horizontal metal track in the matrix (a row of the matrix). In swc-2, however, an output from a processing element has to drive all its horizontal and vertical metal tracks (a row and two columns of the matrix), because the transmission gates connect outputs of the processing elements directly to the inputs of the processing elements they drive. Therefore, the required driving capability for swc-1 is more independent of the matrix size. In other words, a larger transmission gate is needed for bigger switch matrices.

Figures 2.10 and 2.11 illustrate the circuit layout of the two switch cells in 2.5μm double-metal CMOS technology, which were fabricated at Plessey. Figure 2.12 shows the circuit layout of swc-2 in 3μm double-metal CMOS technology which was fabricated at MCE (Micro-Circuits Engineering Ltd). The results from these designs are presented in Chapter 4.

2.4.2. Programming Switch Matrix

The information for switch setting within the switch matrix is provided by users who commit the superchip to a specific system task. The information governing all the switch settings in a switch matrix is here called the *switching pattern* of the matrix. The control circuits are not only responsible for loading the switching pattern, but also help during the testing of the switch matrix. The overall control path of the switch matrix is shown in Figure 2.13. Two decoders are provided for both matrix column and matrix row addressing. Static CMOS NOR gates are adopted here to form both decoders. Each output of the row decoder is connected to the "row load" of all the switch cells sitting on the row which the row decoder output addresses. Similarly, each output of the column decoder is connected to the "column load" of all the switch cells sitting on the column which the column decoder output addresses. The required input address for the control circuits is the binary form of the actual location of the matrix row or column. For example, the input "1010" to the row decoder will address the 10th row in the

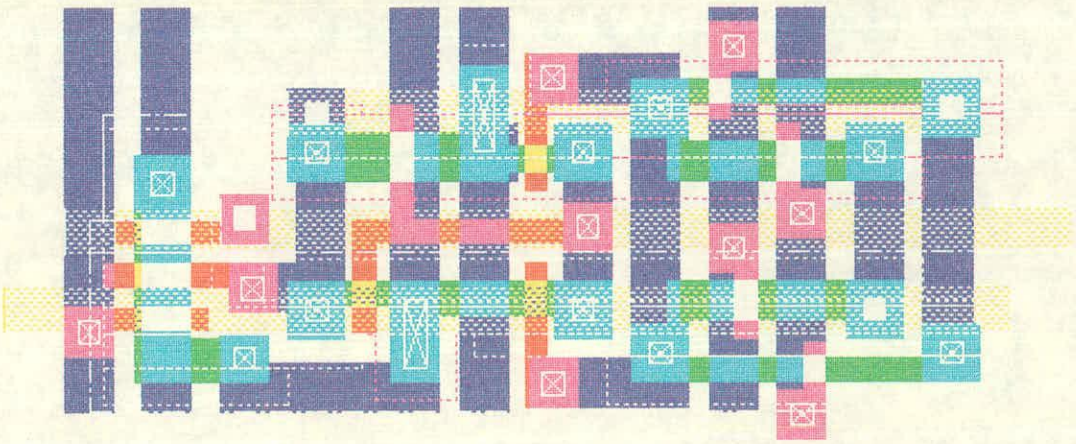

Figure 2.10 Layout of swc-1 (Plessey technology)

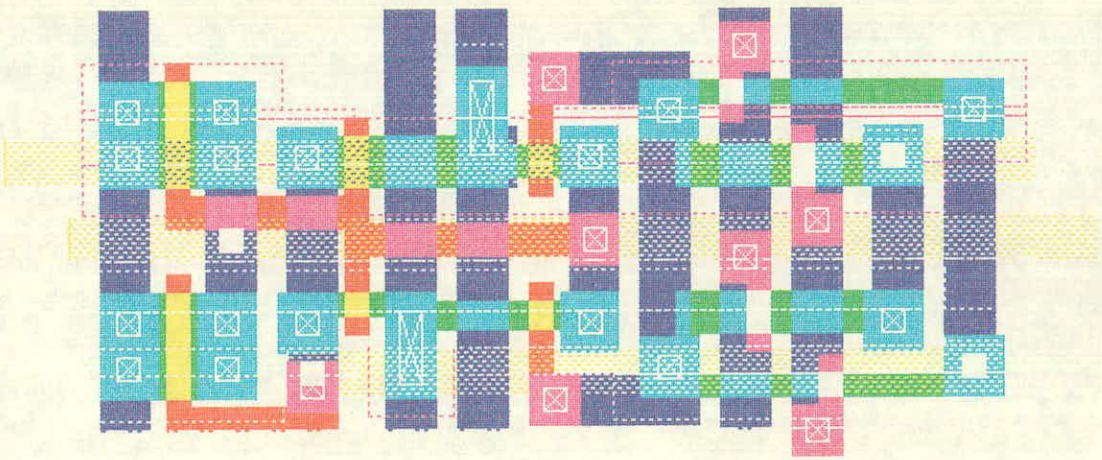

Figure 2.11 Layout of swc-2 (Plessey technology)

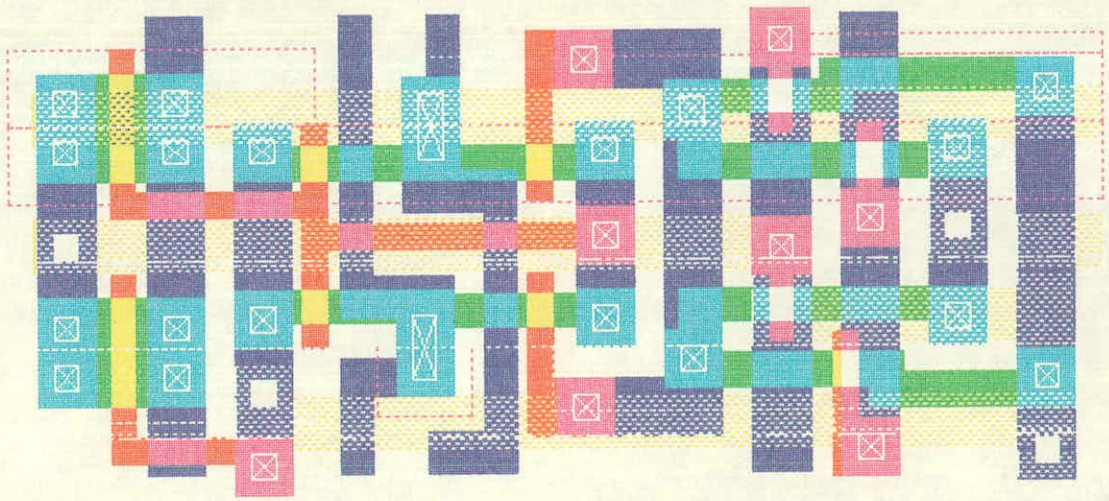

Figure 2.12 Layout of swc-2 (MCE technology)

switch matrix. Prior to loading a switching pattern, the whole matrix is initialised (every switch cell is set to be "off"). To initialise all the switch cells simultaneously, all the outputs of the row decoder should be set to be at a logic zero at the same time. This is realised by decoding an address which is greater than the existing number of total rows in the matrix. If the total number of rows in the matrix is just the value of two to the power of n (where n is the number of inputs to the row decoder), then a spare input should be added to the row decoder. Thus the number of inputs to the row decoder becomes n+1 to ensure that there is at least one address which can set all the outputs from the row decoder to be logic zero at the same time. Such a condition of the row decoder when all its outputs are logic zero, at the same time, is called the *clearing state*. Every output from the column decoder is fed through a two-input NAND gate. This allows all the outputs from the column decoder to be set to logic zero, at the same time, to disable the loading from the row decoder. This is controlled by the signal "Dset" applied to one input of all the NAND gates. This state of the column decoder is here called the *disabled state*.

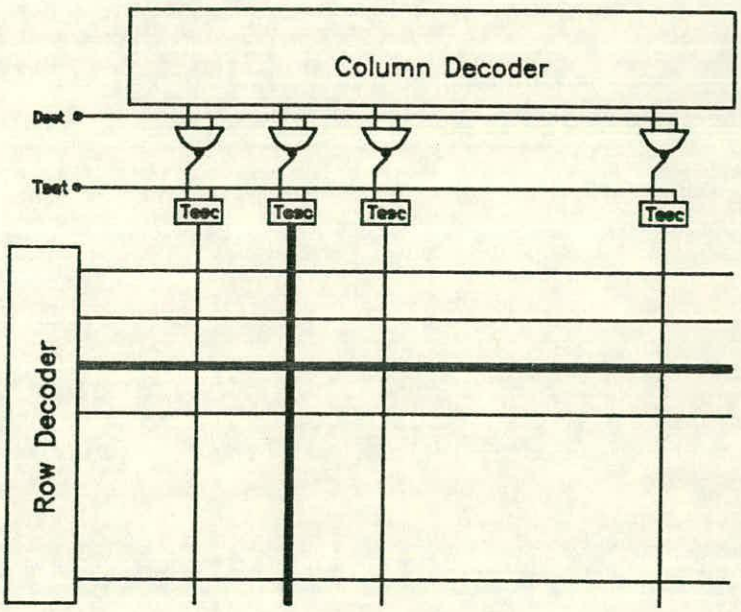

Figure 2.13 Switch matrix control path

The circuit called "Tesc" in Figure 2.14. is an auxiliary circuit for testing the matrix. During test, all the column addressing lines from the column decoder are

expected to be at a logic one, simultaneously, so that all the switch cells on the same matrix row can be addressed and evaluated at the same time. The "Tset" signal sets the operational mode of the test circuit. When it is logic zero, the signal from the NAND gate simply goes through this test circuit to the matrix. Otherwise, the circuit isolates the matrix from the column decoder and sets all the matrix "column load" lines to logic one. The column decoder (including NAND gates and "Tesc" circuits) is said to be in the "*clearing state*" when "Tset" is at logic high.

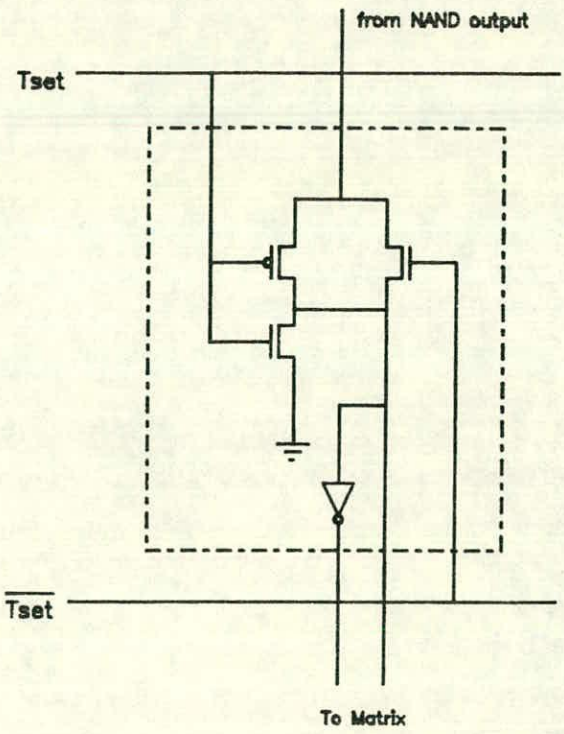

Figure 2.14 "Tesc" circuit diagram

2.4.3. System Operation

System customisation within the superchip is specified and performed by the end user. The whole procedure from the user's system specification to the physical realisation of the system can be divided into two phases, as illustrated in Figure 2.15. The first phase is message generation for system traffic routing. This takes the user's system specification and the test results of both the processing elements and the switch cells as inputs, to produce a set of routing codes (switching pattern) for the switch

matrix. These routing codes are stored in an external EPROM. The test results, showing types of faults and their locations in a superchip, are called the *fault pattern* for any given chip. Defect-tolerance is achieved by connecting working processing elements together using working switch cells in the switch matrix, whilst avoiding defective processing elements and switch cells. Both the routing algorithm and its software implementation will be discussed in detail in Chapter 5. The second phase is system realisation (customisation) which is achieved by loading the switching pattern obtained from the first phase into the switch matrix to realise the interconnections for a specific system.

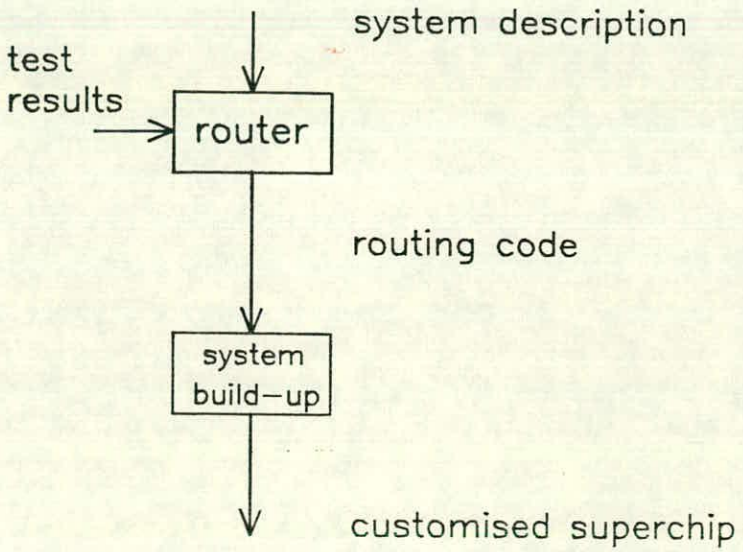

Figure 2.15 System synthesis procedure within the superchip

The switching pattern for the switch matrix is loaded on power-up, or in response to a reset request from an external EPROM. Before actually loading the switching pattern, both the row and column decoder are set to the clearing state to initialise the whole matrix, after which all the switch cells in the matrix are non-activated (non-conducting). Although the total number of switch-settings for the switch matrix is large, the embedded information is much smaller, as at most one switch in any column may be active. Thus we need only to store and load a row address for the active switch cell in each column. Row 0 (or any assigned row) is reserved for "no connection" and will be employed by unused processing elements to inhibit their clock connection. Thus, unused processing elements do not need to be active, eliminating unnecessary power

consumption. This convention leads to the loading arrangement shown in Figure 2.16, in which a counter is used to address the external EPROM. Each address from the counter is also decoded to activate an associated column in the switch matrix. The active switch for that column is defined by the returned EPROM address generated by the router, which is decoded to drive the relevant "row load" line. Whilst loading a switching pattern, "Tset" is kept low. To prevent possible output signal 'spikes' occurring when the inputs of both row and column decoders change from being incorrectly loaded into switch cells, "Dset" should kept low when addresses from the decoders are being changed. Figure 2.17 shows the waveforms for activating a switch cell in a matrix column. Every time "Dset" goes high, an addressing signal from the row decoder is passed on and loaded into a switch cell, so that a switch cell is set to be active (conducting). The input and output signals are bit-serial, and enter and leave the superchip via the switch matrix. In other words, any element in the superchip can only communicate externally through the switch matrix.

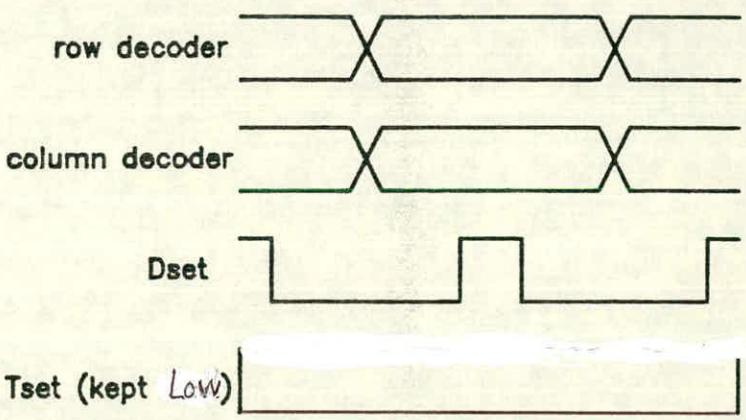

Figure 2.16 A loading scheme for switch matrix

The communication and control conventions for bit-serial systems have been defined by Lyon[90] and Denyer[88]. The lowest level of control is a synchronous bit-clock controlling the flow of all data in the system at bit level. This level of control is called cycle 0, or simple **c0**. One level above this is the cycle indicating the beginning of every new word in a data stream, called **c1** control. At a higher level, there will

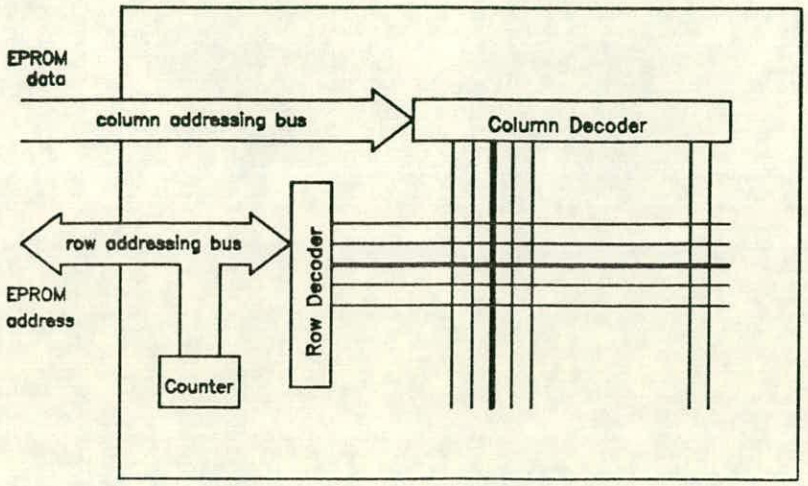

Figure 2.17 Loading signals for activating one switch cell

probably be a need for levels of control indicating the beginning of every frame of words **c2**, and the beginning of a group of frames **c3**, and so on. All these different levels of control signals are generated from an off-chip controller. Figure 2.18 shows a frame of these control signals. The bit-clock control **c0** from the controller is globally distributed to all the processing elements in the superchip. Other levels of control (particularly **c1**) are distributed through a control network in the superchip to adjust various LSB-times of different nodes in the system. The delay elements for such a control network are a part of the processing elements connected to the switch matrix, so that they can be routed (distributed) with correct timing alignment, to any processing element in the superchip.

Synthesizing a system from the superchip can be viewed as a static reconfiguration procedure i.e. the system is configured before operation. However, such a tailored system is also capable of performing dynamic reconfiguration, in which the system configuration can be altered during operation by changing the switch setting in the switch matrix, provided that no communication is required on the connections during the change. In the switch matrix, any connection being reconfigured will not affect communication of other unchanged connections.

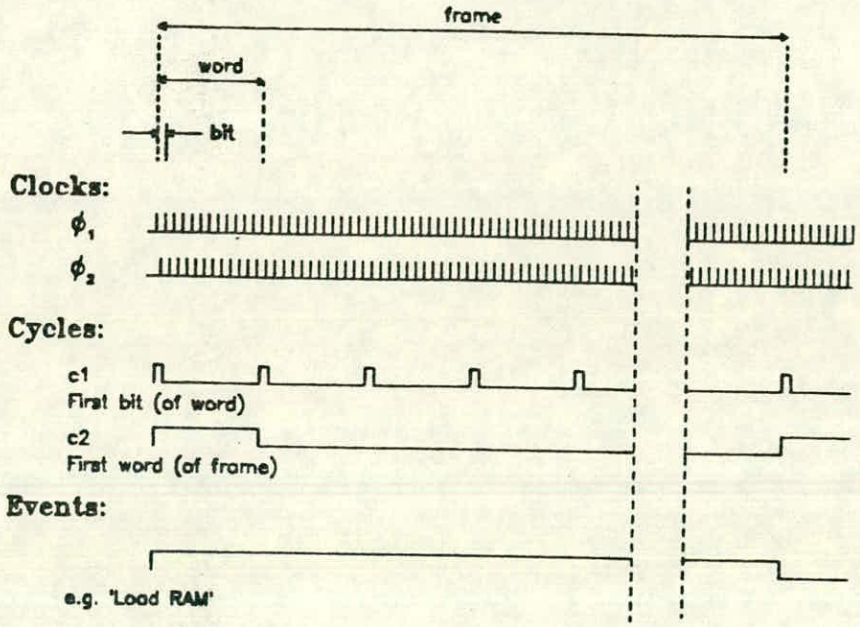

Figure 2.18 A frame of control signals at different levels

2.5. Summary

A system architecture based on integrating a crossbar switching network and a large number of processing elements on a single piece of silicon was presented in this Chapter. This approach enables a high level of integration, and exhibits better performance and lower cost than conventional integration techniques. Some related design issues have been discussed. Such an approach provides defect/fault tolerance and system reconfiguration by message routing in the switch matrix. More numerical and algorithmical representations for such a approach will be illustrated in the following Chapters.

CHAPTER 3

YIELD MODELLING AND REDUNDANCY SELECTION

3.1. Introduction

From the earliest days of integrated circuit manufacture, it has been observed that yield decreases with increase in chip area with some exponential dependence. The impact of this observation to today's VLSI-density chip is the limitation on the chip size for economic production. However, with the advent of defect tolerant circuit design, introduction of hardware redundancy can improve the situation. The question now is one of choosing the right amount of redundancy which is optimal in terms of cost-effectiveness. In this Chapter, discussion will be concentrated on developing a yield model for the superchip, from which an optimal level of redundancy can be derived. Firstly, a brief review of integrated circuit yield modeling is given and, thereafter, these results are applied to the yield modeling of the switch matrix and the superchip.

3.2. Previous Work on Yield Modeling

Integrated circuit yield estimation has been studied since the earliest silicon IC processes were introduced, and many yield models have been proposed. At the outset, the fundamental assumption was that defects occur randomly. In other words, they are distributed uniformly across the whole wafer. If this is true, then the defect distribution can described by the Poisson process

$$P_n = \frac{(D_O A)^n \, e^{-D_O A}}{n!} \tag{3-1}$$

where P_n is the probability of having n defects in the area A, D_O is average defect density (defects/area), A is critical area (or susceptible area[†]), and n are the number of

defects causing initial electrical failure. The yield is obtained by letting n = 0 (note that 0! = 1)

$$Y = e^{-D_0 A} \tag{3-2}$$

and this expression is widely referred to as the so-called Poisson yield model. The dotted line in Figure 3.1 gives the log yield versus total critical area plot (Y-vs-A) for equation (3-2). Unfortunately, this linear relationship (in the log scale for yield) is contrary to integrated circuit manufacturing experience. It is known in practice that experimental log Y-vs-A plots are usually concave upward, as shown by the solid line in Figure 3.1. This fact has led to many modifications of the basic Poisson yield expression.

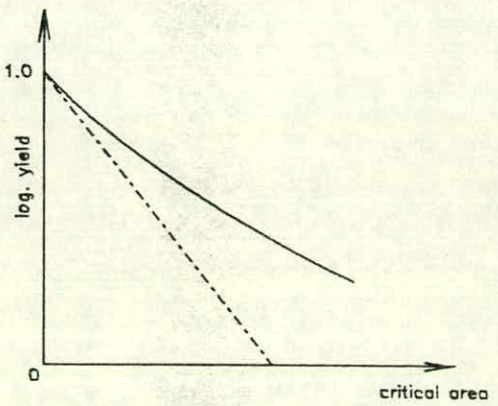

Figure 3.1 Yield vs critical area plot

3.2.1. Murphy's Modification

In 1964 Murphy [91] recognized that the Poisson distribution is not an accurate description of IC yield vs the total critical area of an IC, and tends to give an increasingly pessimistic yield projection for larger areas. This is because the Poisson process assumes that all the defects are randomly, and uniformly distributed over the surface of the silicon wafer. However, in IC manufacturing the defect density varies from chip to chip as well as wafer to wafer. To relate yield prediction given by equation (3-2) with

† critical area is the area within which a single processing defect will cause incorrect logical output of the circuit.

actual manufacturing data, Murphy and some others suggested that the defect density has a long-range non-uniformity expressible by a normalised distribution function (or, in the terminology of probability theory, a *probability density function* f(D)). These are called *mixed Poisson statistics*. The resulting yield expression is given by

$$Y = \int_0^\infty e^{-DA} f(D) dD \tag{3-3}$$

which is known as Murphy's yield model. In this expression, Murphy used both triangular and rectangular distribution functions for f(D). The resulting yield formulae derived from these distributions showed that the yield of larger and more complex chips would be higher than was expected from Poisson statistics. However, according to actual data, the log Y-vs-A curve is still too pessimistic (related curves will be shown later in the summary).

Several other forms of distribution function have also been investigated by Seeds [92], Price [93], Stapper [94,95,96], and Hu [97] etc, in addition to the triangular and rectangular distribution functions adopted by Murphy. Each assumed distribution results in a different form of IC yield model.

3.2.2. Seed's Yield Model

Seeds [92] selected the exponential probability density function:

$$f(D) = \frac{1}{\alpha} e^{-\frac{D}{\alpha}} \tag{3-4}$$

which has the interesting feature that the mean and variance are exactly equal to the average defect density. This distribution leads to a yield expression:

$$Y = \frac{1}{1 + DA} \tag{3-5}$$

which is regarded as a Bose-Einstein distribution. Price arrived at equation (3-5) by basic probabilistic arguments in which all the defects concerned are considered to be indistinguishable. Price favoured the use of Bose-Einstein statistics because they result

in a yield at large values of chip area that is higher than that given by the Boltzmann exponential relation (Poisson statistics). The main argument against using Bose-Einstein statistics is that they assume that the probability of having a defect on a particular chip is the same as the probability of having a defect in the remainder of the silicon wafer, having a much larger area. However, it is felt intuitively that the probability of having a defect should be proportional to its related area.

For a large DA product, the physical meaning of Seed's model is that the fault distribution function is a square function, and the fault density for a (square area) wafer increases linearly across the wafer from 0 to $2q_o$, where q_o is the average number of fault-causing defects per circuit.

3.2.3. Stapper's Model

Stapper suggested [94,95] the use of the gamma function as a probability distribution. The gamma distribution function has the form

$$f(D) = \frac{1}{\Gamma(\alpha)B^{\alpha}} D^{(\alpha - 1)} e^{-\frac{D}{B}} \tag{3-6}$$

where α and B are two empirical parameters. The mean, variance, and σ/u ratio are given by:

$$D = \alpha B \tag{3-7}$$

$$Var\ D = \alpha B^2 \tag{3-8}$$

$$\frac{\sigma}{u} = \frac{1}{\sigma r}\alpha \tag{3-9}$$

The resulting yield calculation can be described by the negative binomial distribution:

$$Y = (1 + \frac{DA}{n})^{-n} \tag{3-10}$$

where

$$n = (\frac{u}{\sigma})^2 \tag{3-11}$$

is a clustering effect coefficient representing the degree of clustering. It is an empirical parameter and usually ranges from 0.5 to 4 in practice. By itself n does not denote the balance between on-wafer defect density variation and wafer-to-wafer defect density variation. However, it is a key clustering parameter and determines the overall yield for chips without redundancy. For situations where the defect density can be assumed constant over a small sub-area, the clustering of defects, either because of interaction among themselves, or because of the hospitality of the geographical characteristics of wafer regions, will tend to result in a higher yield than predicted by the Poisson distribution for a given average defect density. This is because more than one catastrophic defect resides on the same area or sub-area of a chip leaving other chips to have less chance of being defective.

3.2.4. Hu's Model

By employing different assumptions for the distribution functions in the original Murphy yield expression (equation (3-3)) by Seed, Stapper and Murphy, Hu has proved mathematically [98] three properties of the yield model, expressed by equation (3-3). Firstly, Murphy's model is always concave upwards. Secondly, the initial slope of Murphy's model is the same as that of a Poisson curve with the same defect density. Thirdly, the lower bound for Murphy's model is the Poisson yield model.

Hu derived [97] a yield expression from very basic probabilistic considerations, which turned out to be a binomial yield model:

$$Y = (1 - \frac{A}{S})^n \tag{3-12}$$

where n is equal to DS, A is total critical area of a chip, D is defect density distributed randomly over an area S ($A \in S$) which may be a part of or the whole of a silicon wafer, so that

$$Y = (1 - \frac{A}{S})^{\frac{S}{A}DA} \tag{3-13}$$

$$= e^{-DA} \qquad \text{for } \textit{very large } S,\ (\frac{A}{S} \to 0).$$

This equation is the Poisson distribution equation for zero defects, and DA is the statistical expectation. With this model, Hu concluded that it is not appropriate to take account of the phenomenon of non-homogeneous defect populations (defect clustering) by integration with a defect distribution function, either in the geometrical space or in the density space. Instead, the chip should be partitioned into many sub-areas, and the overall yield should be obtained by partitioning sub-populations of defects in these sub-areas. Hu noted that, although almost all the modifications of the Poisson distribution introduced by integration with a density distribution function show some concavity in the log yield vs area plot, these modifications are not based on rigorous mathematical procedures. The yield data predicted by these modified yield models agree with practical data, but the extrapolated yield from these models may not be reliable.

Another interesting conclusion that Hu pointed out [97] is that the applicability of the simple Poisson distribution increases with the area of an IC chip. If the size of a given IC chip increases to one quarter of a wafer, then the simple Poisson distribution would become more accurate, regardless of the non-uniformity in defect density. In other words, if the defect distribution function f(D) is applied, then it must involve chip area. Thus variations in defect density tend to be averaged out as the chip size becomes large with respect to the wafer.

As Stapper commented [99], the difficulty with Hu's yield model is that it requires the ratio of critical area to chip area of each partitioned sub-area to be identical. As far as Poisson statistics are concerned, it is not necessary to use m sections from different chips and wafers to allow the binomial statistics to approach Poisson statistics, by considering the chip area in the defect distribution function f(D). The Poisson model holds for each chip individually, but λ, the average number of faults per chip can, and often does, vary from chip to chip, even on a single wafer. The mixed Poisson statistics are, therefore, a valid method for modelling chip-to-chip variations in defect or fault density.

Some experimental data collected recently from several IBM manufacturing lines for some large area chips suggested [100,101] that the yield predicted by the negative binomial yield model shows good agreement with measured data, which indicates the possible viability of this yield model for large area ICs. The predicted data seem to be strongly dependent on the clustering parameter n, which indicates the importance of the clustering effect on yield modeling, particularly for large integrated circuits.

3.2.5. Paz and Lawson's Yield Model

Paz and Lawson suggested [102] the following yield model:

$$Y = Y_O e^{-AD} \tag{3-14}$$

where Y_O is a gross cluster yield when the whole chip is partitioned into several smaller areas, and e^{-AD} represents the simple Poisson random defect yield. The values of Y_O and D were determined with a linear regression technique using

$$lnY = -AD + \ln Y_O \tag{3-15}$$

which was performed for each region on each wafer. Results confirmed that the wafer to wafer defect densities could be modelled by a gamma distribution. The defect density in the outer region of a wafer was found to be high and, therefore, the gross cluster yield Y_O was lower there. Furthermore, it appeared that the wafer to wafer distribution of Y_O could be modelled with a beta distribution.

3.2.6. Moore's Model

Independent of the mixed Poisson statistics, an empirical expression describing integrated-circuit probe yield experience at Intel [103] was reported by Moore:

$$Y = e^{-(\frac{A}{A_{OM}})}. \tag{3-16}$$

where, A is the IC area, and A_{OM} is reference area (which can be regarded as the IC area that leads to a probe yield of $\frac{1}{e}$ or 37%).

3.2.7. Dingwall's model

According to actual data and experience [104] at TRW, Dingwall suggested [105] a probe yield model with a value $\alpha = 3$ in equation (3-10).

$$Y = (1 + \frac{\bar{D}}{3}\frac{A}{A_{OD}})^{-3} \tag{3-17}$$

where, $\bar{D}$ is a dimensionless mean defect density (i.e. the average number of point defects per IC), A is the IC area as before, and A_{OD} is a reference area which is the IC size for a given process that leads to a probe yield of 42%, when the mean defect density $\bar{D}$ is one defect per IC. This empirically modified negative binomial yield model has proved reliable at TRW over many years.

3.2.8. Warner's Composite Model

This model [106,107] combines both Moore's and Dingwall's models, making an assumption that half the aggregate area of all the areas considered for yield evaluation is homogeneous (in the sense that the point defects therein constitute a single binomial-Poisson population) and characterised by a low value of defect density. Warner introduced a window method, similar to the concept of yield scaling described by Stapper. The window method concludes a linear relationship between log yield and area. The statistical meaning of such a linear result is that any catastrophic defects involved constitute a homogeneous Poisson population, such that:

$$Y = Y_O^{\frac{A}{A_O}} \tag{3-18}$$

where Y_O is the yield at the area A_O within the homogeneous region.

3.2.9. The Clustering Effect

Stapper recently described [108] the effects of on-wafer or wafer-to-wafer defect density variations on IC defect distributions. His result of yield vs area plot for cluster yield is presented here in Figure 3.2. The two end points are those measured in [109]. The flat portion on the left indicates the constant Y_O range for small chips identified in

Paz and Lawson's yield model expression. The extent of the flatness depends entirely on the areas of the smallest clusters that are considered. The curvature of the yield plot for larger chips depends on the size distribution of the clusters. If there are a large number of small clusters, then the roll-off will be sharp. For a distribution with more large clusters the yield curve will fall off more slowly. From this curve, it is clear that success in increasing chip area (therefore, increasing chip complexity) towards the whole wafer size will depend on the scale of clustering for a given technology. For large wafer-to-wafer variation of defect density, the yield of larger chips or a whole wafer can be improved, due to the large population of defects clustered on a limited number of wafers of a batch. On the other hand, on-wafer variation of defect density (defect clusters within the chip or wafer area) increases the need for redundant circuits more than that predicted by Poisson statistics and simple Poisson yield models and, therefore, results in lower yield for chips on the wafer.

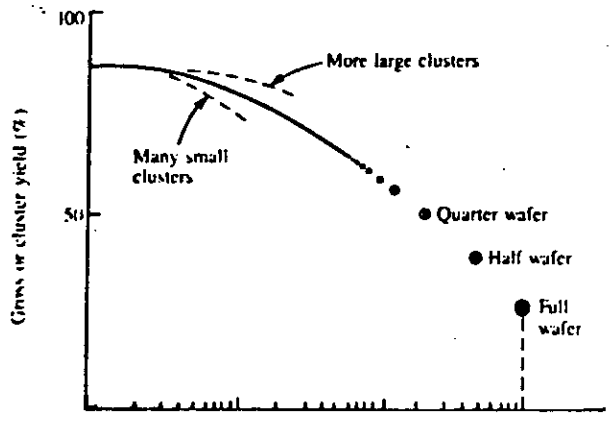

Figure 3.2 Cluster yield vs area

3.2.10. Summary of The Yield Models

Since Murphy first introduced the defect distribution function, considerable investigations have been carried out to find a suitable distribution function to predict the manufacturing yield of chips, over a large range of chip areas. If the Poisson yield model is considered to be included in the Murphy's yield expression (mixed Poisson statistics), then its defect density distribution function is a delta function at its average defect density. Figure 3.3 is a composite of several defect density distribution functions previously proposed. The distribution functions given in Figure 3.3 have been normalised to the average defect density D_o, given in equations (3-1) and (3-2). Figure 3.4 is

the log Y-vs-A curves related each equivalent distribution function in Figure 3.3. The Poisson yield curve in Figure 3.4 corresponds to the delta distribution function labelled A in Figure 3.3. An exponential probability distribution function, labelled C in Figure 3.3, leads to Seed's model, which is labelled C in Figure 3.4. The curve B in Figure 3.3 is a gamma function proposed by Stapper, which gives the generalized negative binomial yield model indicated by curve B in Figure 3.4. In fact, curve B varies with the parameter n; when $n < 1$, and lies below the curve C, otherwise it resides above C. Curves D and E in Figure 3.3 are for rectangular and triangular functions, explored initially by Murphy to derive the yield curves D and E in Figure 3.4. Unfortunately, these two expressions have not found extensive use in subsequent work, primarily because of the low fault distribution values for the ratio of standard deviation to the mean, σ/u. Data subsequently analysed by Seeds [92], Moore [103], Stapper [94,109,95,20], and Paz and Lawson [102] suggested higher values for this ratio, leading to even more optimistic yields for large chips than Murphy had anticipated. It should be noted that all the yield curves shown in the diagram broadly fit today's typical VLSI chip size (which lies within several tens of square millimetres) for current silicon technology. Therefore, random defects over a chip are still the predominant factor in determining yield values. In the case of a very large chip, the effect of defect clusters may not be neglected in the yield model. Although the clustering parameter has been introduced in the negative binomial model (equation (3-10)), for large chips or (sub) wafer scale integration, there still remains a question about its applicability, particularly when the degree of clustering is small. To take global clustering into account, the introduction of the cluster yield Y_o in Paz and Lawson's model (equation (3-14)) makes it possible not only to separate the cluster yield from the yield caused by the remaining random defects over the whole area, but also to correct the yield curve concave-downwards from the yield models for the mixed Poisson statistics, in the case of large chip or the whole wafer.

Most of the yield models discussed here are derived from either Poisson statistics or mixed Poisson statistics. Therefore, there are some inter-relationships amongst these models. Assuming a gamma distribution for the defect distribution function given in equation (3-3), a negative binomial yield model can be derived as shown in equation (3-10). If $n \to \infty$, this model becomes the Poisson yield model; whereas, if $n = 1$, then it becomes equation (3-5), which is also known as Seed's equation.

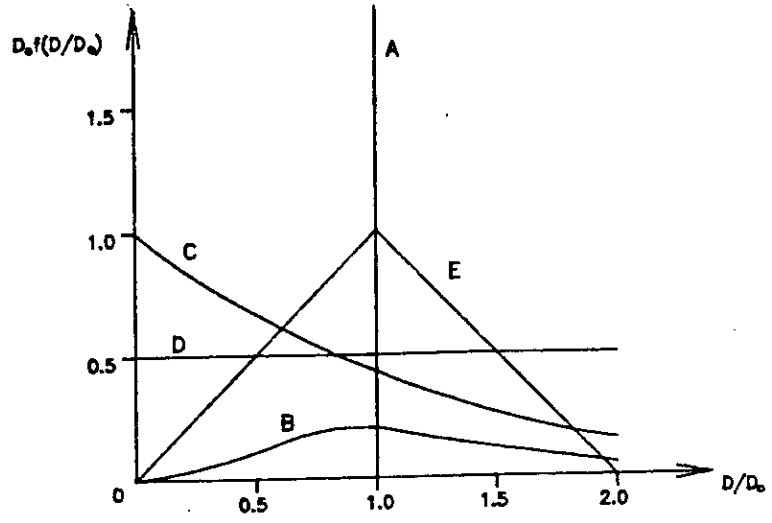

Figure 3.3 Defect density distribution functions. (A) Delta function leading to Poisson yield model; (B) Gamma function leading to the negative binomial model (here particular values has been chosen for the two adjustable parametres); (C) Exponential function leading to Seed's model; (D) Rectangular function used by Murphy; (E) Triangular function used by Murphy.

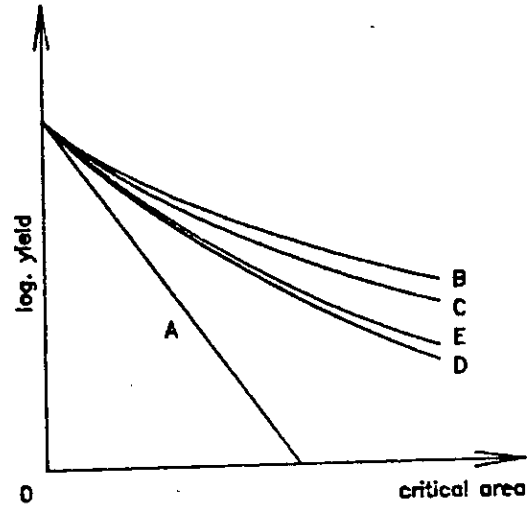

Figure 3.4 Yield curves resulting from several different types of defect density functions. (A) Poisson yield curve from the Delta function; (B) Negative binomial yield curve from the Gamma function; (C) Seed's curve from the exponential function; (D) Yield curve from the rectangular function; (E) Yield curve from the triangular function.

It should also be pointed out that most of the yield models are derived from equation (3-3) by choosing a defect density distribution function to best fit the manufacturing yield data. These data are usually collected on site for the normal VLSI chip size ($< 100\ mm^2$). Each of these yield models often suit a particular silicon processing site. It is very difficult, and probably not sensible at all, to search for a common yield model. Furthermore, extrapolation of these yield models for larger chip sizes may not be reliable, although some recent research [110,100] has shown the validity for large memory chips. However, more information is still needed to validate its correctness.

3.3. Superchip Yield Modelling

The entire yield modelling process for the superchip architecture, introduced in Chapter 2, can be divided into two parts: switch matrix yield modelling and processing element yield modelling. Processing element yield modelling is largely dependent upon the processing elements themselves. Depending upon their complexity (size), a suitable yield model reviewed in Section 3.2 can be chosen. Yield calculation of all the processing elements is a simple probability problem. The most involved aspect is yield modelling of the crossbar switch matrix. Most of the discussion in this Chapter relates to developing a functional yield model for the switch matrix. As a basis for the functional yield model, yield figures for a single switch cell are extremely important. These yield figures will be derived from the defect sensitive areas (critical area) within the two different switch cells. Then, with the yield figures available for the switch cell, a functional yield model for each type of switch matrix can be developed. Redundancy selection is determined by the measure of cost-effectiveness, concerning the yield improvement and the cost of hardware overhead.

3.3.1. Fault Analysis of the Switch Cells

Logic faults in integrated circuits are usually attributable to the following sources: physical defects in active devices such as transistors, and physical defects in passive devices such as interconnect lines, and the contacts and vias between them. Extensive analysis of such failure mechanisms have been reported [19,111,112,113]. At circuit and logic level, in which the transistor is the basic device, physical defects can cause open and/or short-circuit connections to either the source or the drain structures of each MOS transistor, or a possible missing connection to the gate of an MOS transistor. Two

classical logic faults namely, stuck-at-one (SA1) and stuck-at-zero (SA0), are attributed to a low-impedance, short-circuit, to VSS or VDD, respectively, either through a faulty transistor or a shorted interconnection line. For CMOS circuits, these two logic faults are not sufficient. Other logic faults, mainly stuck-open (S_OP) faults, introduce undesired high impedances at certain nodes which the classical stuck-at fault can not model. This type of fault is usually called a *non-classical* fault. To include such faults within the fault model, the use of gated latches has been suggested [114]. However, this approach results in a very complicated fault model even for very simple logic gates, such as a two-input NOR, or CMOS transmission gates, etc. For CMOS circuits, particularly static CMOS circuits, all the above SA1, SA0, and stuck-open faulty nodes can be modelled by, or attributed to, a set of transistors with their sources and drains permanently stuck-on (S_ON) or stuck-open (S_OP), thereby losing any gate control. For the switch cell circuits presented Figures 2.7 and 2.9, probably the more practical way to analyse the possible faulty behaviour of the circuits is to consider each transistor S_OP and S_ON and its contribution to the overall faulty behaviour of the circuit. This analysis must include not only the classical, but also the non-classical logic faults. Open- and short-circuits in the interconnection lines are too layout-dependent, so that they will not be taken into the consideration at the logic level. They will be considered when the defect-sensitive (critical) area of the switch cell circuits is calculated later for the switch cell yield evaluation.

Figures 3.5 and 3.6 show two switch cells again, which have already been given in Figures 2.7 and 2.9 previously, but now at a logic gate level. Both swc-1 and swc-2 have a CMOS latch controlled by "column load", which allows the switch-setting message, represented at "row load", to be loaded and then stored in the latch. A NAND gate performs a transistor switch in swc-1, while a CMOS transmission gate acts as a transistor switch in swc-2, both of which are controlled by the output of the latch for its on or off state. All the following presentations are based on extensive fault simulations, in which a single fault is mainly assumed. Some multiple faults are also analysed for particular cases. However, the analysis of random multiple faults is too complicated and certainly beyond the scope of the discussion at this stage. In the following treatment, all the transistors in two switch cells are named by their labels given in Figures 2.7 and 2.9, and the analysis is carried out part by part within each switch cell.

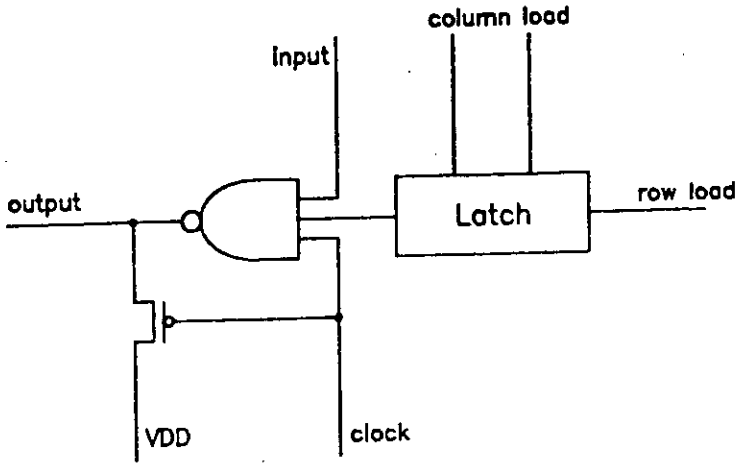

Figure 3.5 Gate level diagram of swc-1

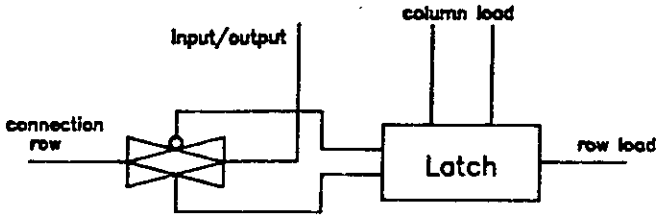

Figure 3.6 Gate level diagram of swc-2

Dynamic Switch of swc-1 (NAND gate):

Any of the three transistors of T9-T11 in Figure 2.7 being S_OP, will not discharge the output node. This is equivalent to the switch being permanently off. When T9 is S_ON, the switch behaves as though it is permanently on. T10 being S_ON may cause an incorrect output, if the cell is intended to be on. The incorrect output is irregular in terms of a switch being permanently "on" or "off". This fault can be bypassed by not being activated. It is shown later in the treatment of routing algorithm that such a cell can be considered the same as being S_OP. T11 being S_ON will not cause any logic fault.

Transmission Gate

The output of the transmission gate being SA1 or SA0 may be caused by short-circuits between the output and VDD or VSS. For the structure of the duplex matrix,

this can cause an entire matrix column or row to be SA1 or SA0. As seen later, this situation can not be tolerated with the scope of the switch matrix itself. Any of these faults will straightforwardly prevent the use of the particular matrix column, or matrix row, where the stuck-at fault resides. These two faults can be viewed as the classical logical faults (stuck-at faults).

Besides the two classical logic faults, two non-classical logic faults (S_OP, S_ON) are both possible in a transmission gate. Either of the two transistors in the S_ON state cause the switch to be permanently on. This may result from a defect in the transistor itself or a faulty latch causing its output SA1. Similarly, when both transistors are S_OP, the switch is permanently off. The switch still works when only one transistor in the pair is logically S_OP, but at lower speed. In the crossbar matrix, the fanout for a switch is usually large, so that the driving capability of such a switch is crucial to the system performance. Fault simulation for such a case shows that there is a very long transmission delay and an intolerable drop in output voltage. Therefore, under this situation, the switch is regarded as "off".

CMOS Latch:

As shown in the circuit and logic diagrams of the switch cells (Figure 2.7 and 2.9), the latch can be divided into two parts: a CMOS pass gate controlled by "column load", and a CMOS static latch with its input controlled by the pass gate. The two series inverters with positive feedback are capable of storing the switching message from the pass gate.

The S_OP and S_ON faults in an inverter cause three possible incorrect outputs: SA0, SA1, or S_OP. The two classical logic faults are caused by the transistor being shorted to VSS or VDD respectively. S_OP may be caused by either or both transistors being open to VSS and/or VDD. When S_OP occurs, the inverter output logic value will eventually drop to logic "0", due to the inevitable leakage current. Consequently, this is equivalent to the SA0 fault at the output. Both transistors being shorted to VSS and VDD is not a legitimate fault here, because it represents logical inconsistency. In fact, the output value in this case lies somewhere between VDD and VSS, determined by the impedance ratio of both 'pull-up' and 'pull-down' transistors. Therefore, faults in a CMOS inverter exhibit the two classical logic faults, SA1 and SA0. For the two series inverters, the faulty outputs are the same as a single inverter. As a contribution to the

switch cell, they (SA1, or SA0) will force the transmission gate or the transistor switch (NAND) to be permanently "on" or "off", respectively.

The CMOS pass gate can be further divided into two paths: one path consists of T1 and T4, the other consists of T2 and T3. For either path, when there is only one transistor being S_OP, the circuit can still pass signals, except that the loading speed is slightly slower. This does not affect the system performance so far as the static reconfiguration and defect tolerance is concerned. However, the cell having such a fault may be considered faulty for its lower speed in switch-setting when the matrix is used for dynamic reconfiguration purposes. The fault simulation of this fault showed that the reduction of the loading speed is reasonably small. If the two transistors in either path are both S_OP, then the cell is faulty. This is clear when the open path is actually the path which is to pass the signal from the "row load" (T2 and T3). If the open path is the path of the positive feedback (T1 and T4), then the switching message can be written into the cell. However, when the path is changed afterwards to the feedback path (referring to the loading scheme in Chapter 2), the open path on the feedback route results in a floating input node to the latch. Such a floating capacitive node, when charged, will eventually leak through any parasitic resistance present, causing the latch output to approach logic "0". The switching information stored in the latch is also very vulnerable to noise. When the S_ON fault occurs in the path which passes the signal from the "row load", the cell will be faulty, because the switching setting will be overwritten when the load signal at the "row load" (logic "1" to turn a switch "on") disappears. However, when the S_ON occurs in the feedback path, there is no faulty condition exhibited at the output.

From the above analysis, various faults in the swc-1 and swc-2 mainly cause the switch of the cell to be permanently "on" or "off". Therefore, the later discussions will concentrate on how to tolerate these faults in the switch matrix.

3.3.2. Switch Cell Yield Modelling

Switch cell yield modelling is at the lowest level in the functional yield modelling for the switch matrix. The yield value is related to the geometry of the switch cell layout. According to the fault analysis given earlier, and the actual layout given in Figures 2.10, 2.11, and 2.12, the critical area for each switch cell can be evaluated.

The critical area calculation for a switch cell could be involved, because, strictly speaking, the critical area varies with defect size†. The observation made by IBM over a long period of study suggested a distribution curve falling off with the increase of defect size, according to an inverse-cube relationship. During integrated circuit manufacture, defects introduced at each processing step may be considered in either of two categories: first, very small defects (point defects) which often occur in chip insulators. These are also known as dielectric pinholes. The second category includes defects which have sizes comparable to those of VLSI pattern feature size. All photolithographic missing patterns or extra patterns belong to this category. Intuitively, the critical area of a dielectric pinhole is the overlap region between two conductors, which cross each other. Defects falling outside this area cannot cause logic errors (short circuits). The critical area of pinholes for most design may be determined readily as the total overlap area between conducting patterns at different photolithographic levels (metal layer, polysilicon layer, diffusion layer, etc). The critical area of photolithographic defects is more involved. Most discussions so far have concentrated on the special case that a length of track, l, is much larger than its width, d. With the inverse-cube defect size distribution, the average critical area of open circuit for a long conductive line is [22]

$$\bar{A} = l\frac{x_0^2}{2d} \tag{3-19}$$

Similarly, for the case of short circuits between two very long conductors, the average critical area $\bar{A}$ is

$$\bar{A} = l\frac{x_0^2}{2s} \tag{3-20}$$

where s is the separation between two conductors and x_0 is the most frequent defect size in the IC process.

† For some type of defects, such as missing patterns or extra patterns of metal track, whether they are catastrophic will depend upon their size. For example, a 2μm wide missing pattern in a metal track 5μm wide will not cause any logic fault, even though it does reside in the metal track.

According to the above principles for calculating critical area together with the actual layout of the two switch cells in Figure 2.10 - 2.12, the overall critical area for each switch cell can be calculated. The critical area for pinholes in swc-1 and swc-2 are the same, being 970 μm^2. The average critical area for photolithographic missing or extra patterns depends on the value of x_0. Unfortunately, no values for x_0 were available for the process on which the prototype chips were fabricated. However, Marsh suggested [116] a square-count model for calculating the critical area, where the critical area is defined as the minimum dimension specified in the design rules. This simplification turns out rather more pessimistic than the final resulting yield data, because it has been shown [117,115] that the total critical area is less than the total pattern area and, thus, yield projections based upon the pattern area rather than the critical area will tend to be pessimistic. Because of the lack of defect size distribution data, this simplification is used to estimate the critical areas in the switch cells. The resulting areas are 3318μm^2 and 3296μm^2, respectively, for swc-1 and swc-2, making the total critical areas for swc-1 4288μm^2 and for swc-2 4266μm^2†.

The critical areas of both switch cells are very small, so that the choice of yield model is open. According to Figure 3.4, and one of the properties of the Murphy yield model expression in equation (3-3), all the models derived from the Murphy's expression with different defect density distribution functions will have the same initial slope as the Poisson model, thus giving roughly the same yield value for circuits with very small critical area. Therefore, the Poisson model is used to calculate the yield for the switch cells. Table 3.1 shows the yield values under different defect densities.

3.3.3. Yield Model for the Normal Matrix

Yield Model

According to the normal matrix structure shown in Figure 2.3, in order to make an interconnection, one switch cell in an input column of a processing element has to be set active (conducting), as indicated by the bold line in Figure 2.3. Interconnection between an output of a processing element and a matrix row is hard-wired. Moreover,

† The critical area for swc-2 is obtained from the layout designed in Plessey design rules. However, the critical area of the other swc-2 design in MCE design rules is very close to this figure.

Table 3.1 Expected Yield of the Switch Cells					
Defect Density(cm^{-2})	1	2	3	4	5
swc-1 (%)	99.9957	99.9914	99.9872	99.9824	99.9786
swc-2 (%)	99.9957	99.9915	99.9873	99.9830	99.9787
Defect Density(cm^{-2})	7	10	15	20	25
swc-1 (%)	99.97	99.9572	99.9358	99.9144	99.893
swc-2 (%)	99.97	99.9574	99.9360	99.9148	99.8936

every interconnect route has to pass several switch cells horizontally and vertically. If some faults in these cells cause an unwanted inter-cell short circuit between their equivalent rows and columns, then the attempt to route this interconnection route has to be abandoned.

The yield of the normal matrix is considered as the probability of having N successful interconnection routes in a N×N matrix. Here N is the number interconnections required by the system. Let y_s be the yield of a single switch cell resulting from the above Section. Suppose that a switch cell has the same probability of permanent short-circuit and open-circuit between its input and output, y_b. Then

$$y_b = 1 - \frac{1 + y_s}{2} \tag{3-21}$$

Statistically, the average number of non-activated switch cells passed by one interconnection route is N vertically and $\frac{N}{2}$ horizontally. Let the yield loss caused by photolithographic defects in the output metal tracks of the processing elements right across the switch matrix be y_t. Then the probability of successfully embedding a single interconnection route is given by

$$y_{sin} = y_s y_t (1 - y_b)^{N + \frac{N}{2}}$$

$$= y_s y_t (1 - y_b)^{\frac{3N}{2}} \tag{3-22}$$

The yield of whole switch matrix without redundancy is

$$Y_o = (y_{sin})^N \tag{3-23}$$

Supposing, further, that R redundant columns and R redundant rows are included in the matrix to make N interconnection routes within this (N+R) by (N+R) switch matrix. Then the total yield of whole switch matrix with these R redundancies is given by†

$$Y_R = \sum_{i=0}^{R} C_{N+R}^{N+i} (y_{sin})^{N+i} (1 - y_{sin})^{R-i} \tag{3-24}$$

The relationship between yield and the size of the matrix with various on-chip redundancies, using this equation, is plotted in Figure 3.7. Figure 3.8 shows the relationship between yield and matrix size without any redundancy and under various defect density conditions. In Figure 3.9 and 3.10, the relationships between yield and the amount of redundancy for various defect densities are given.

From Figures 3.7 to 3.10, several observations can be made:

- Yield decreases with the increase in matrix size. Beyond a certain point, it will decrease very rapidly (nearly exponentially). Adding more redundancy will shift this point along the matrix-size axis, therefore indicating better yield values for larger matrices.

- The yield curves saturate after a certain amount of redundancy. This shows that, after a certain level, adding more redundancy is not beneficial.

† Any reader who is not familiar with probability theory, and the basic mathematic of permutations and combinations, is advised to consult Appendix A.

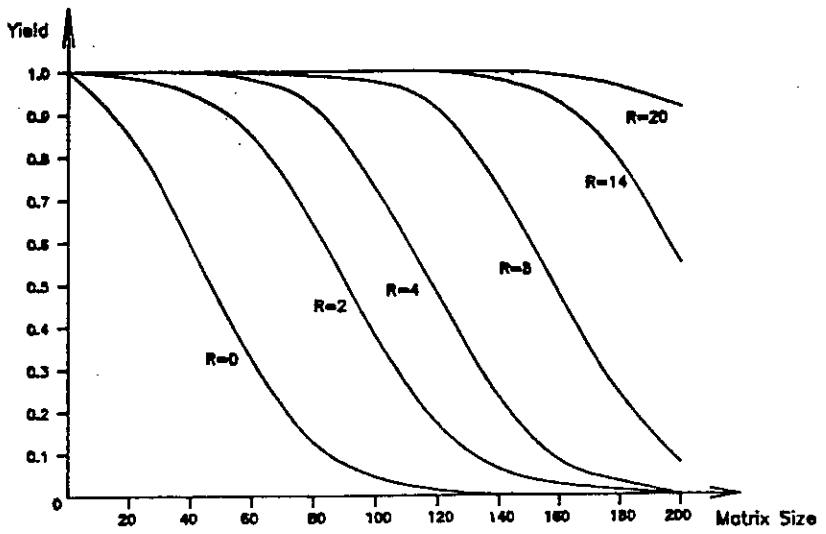

Figure 3.7 Yield vs matrix size plot with various redundancies (if the reading on X-axis is n, then the size of the normal matrix is n×n).

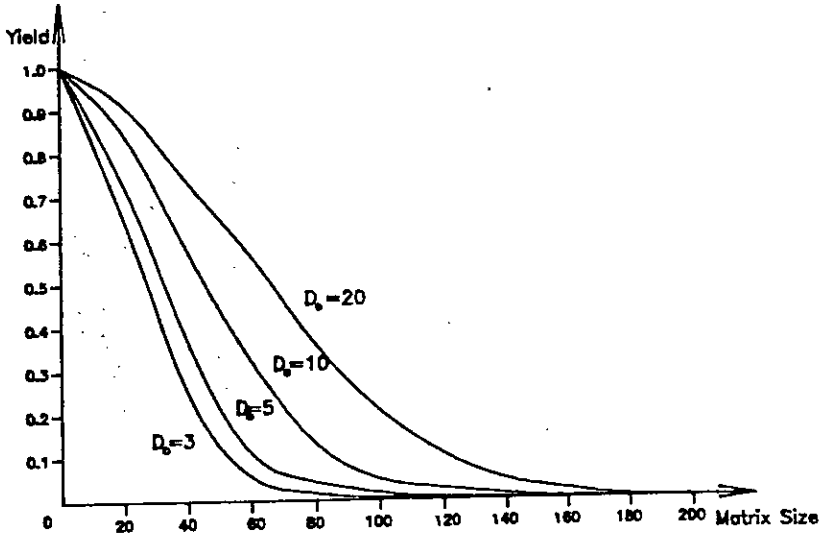

Figure 3.8 Yield vs matrix size plot without redundancy, but with various defect densities, Do (cm^{-2}).

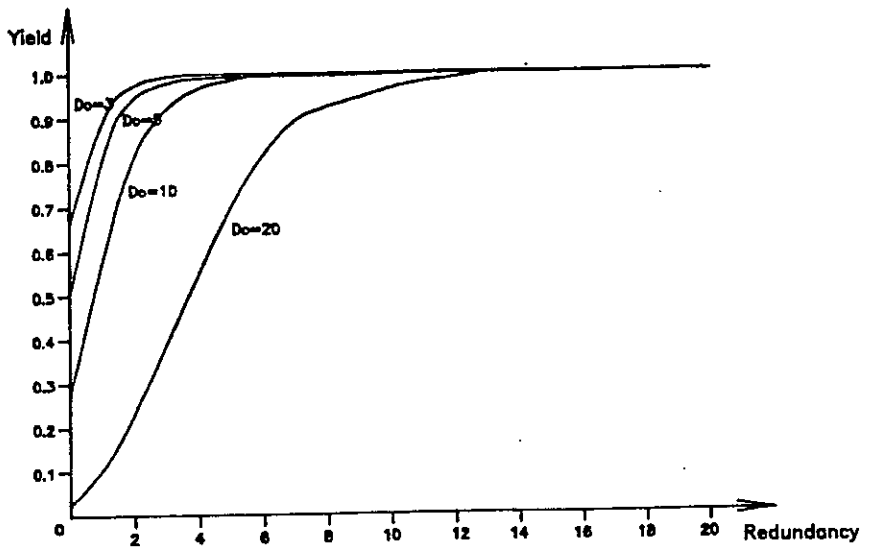

Figure 3.9 Yield vs redundancy plot with various defect densities (the matrix size is 64×64).

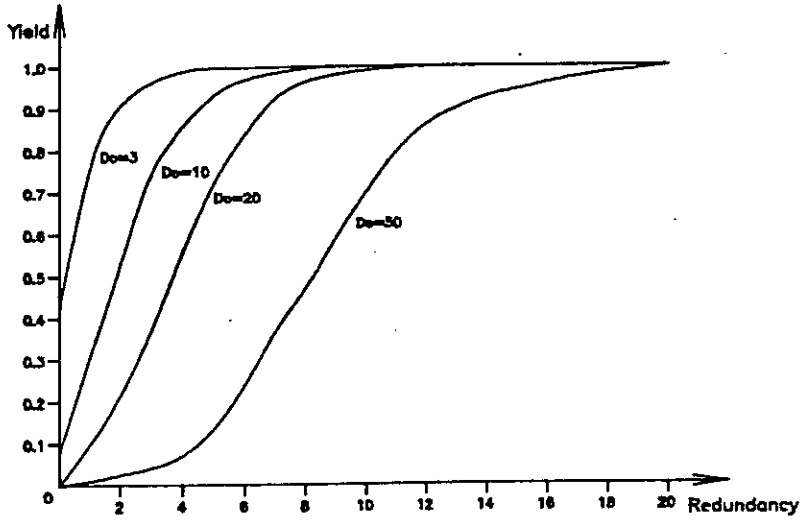

Figure 3.10 Yield vs redundancy plot with various defect densities (the matrix size is 160×160).

- The increase in yield is greatest amongst the chips with lowest initial yield (without redundancy). Therefore, a low yielding chip can obtain more benefits from redundancy in design.

- To obtain the same yield, a larger matrix needs more redundancy if the defect density remains the same.

Cost of Redundancy

Adding redundancy into a system can improve its yield. However, as shown before, adding more and more redundancy will not necessarily exhibit more benefit. For a chip which employs redundancy in design, let Δ be the relative increase in the area due to hardware redundancy, i.e. Δ indicates the hardware overhead, where

$$\Delta = \frac{\textit{chip area with redundancy } A_R}{\textit{chip area without redundancy } A_O} \tag{3-25}$$

It is useful here to introduce a "figure of merit" for the normal matrix, FM_n, indicating the benefit that adding a certain amount of redundancy can achieve. This can be defined as:

$$FM_n = \frac{A_O}{A_R} \times \frac{Y_R}{Y_O} = \frac{1}{\Delta} \times \frac{Y_R}{Y_O} \tag{3-26}$$

When $FM_n > 1$, the cost of a chip with redundancy will be lower than that without redundancy. $FM_n = 1$ is the boundary case.

In Figure 3.11, the figure of merit, FM_n, is plotted for various defect densities and redundancy for a 60 by 60 normal switch matrix. Figure 3.12 gives the same parameter, FM_n, but for different matrix sizes under a defect density $D_O = 10$. The same conclusions may be deduced from the these curves as make in last Section: *the lower the chip yield, the more benefit there will be from redundancy in design.*

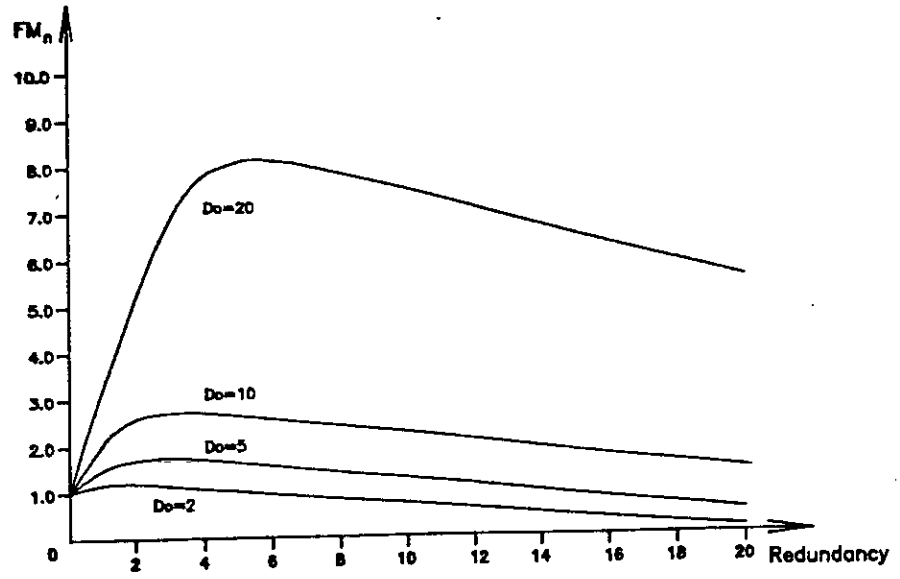

Figure 3.11 FM_n vs redundancy plot with various defect densities (the matrix size is 60×60).

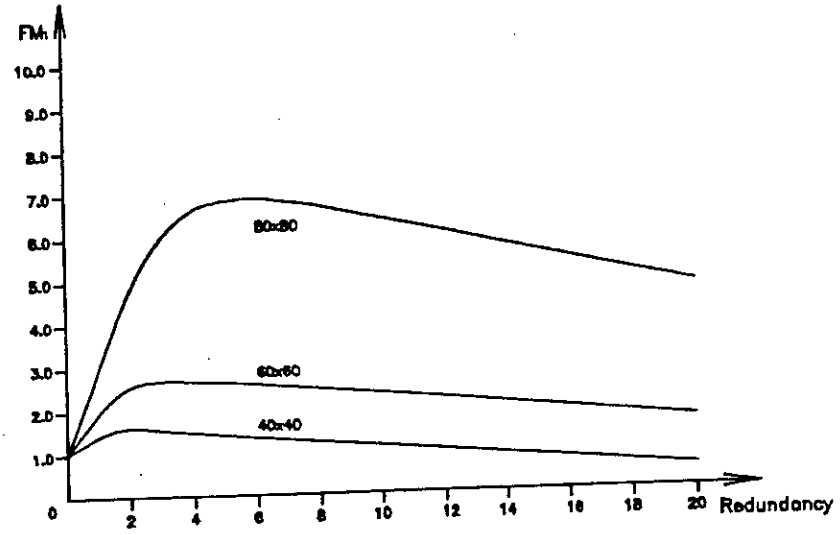

Figure 3.12 FM_n vs redundancy plot with various matrix sizes (defect density is 10 cm^{-2}).

3.3.4. Yield Model for the Duplex Matrix

Yield Model

Unlike the case of the normal matrix, for the duplex matrix shown in Figure 2.5 the outputs of each of the processing elements are not hard-wired to the matrix rows. Instead, through a chain of switch cells, the output can reach any horizontal row in the matrix. Therefore the duplex matrix is potentially more defect/fault tolerant. To make a connection, two switch cells have to be activated. As for the normal matrix case, all the non-activated switch cells through which an interconnection route passes must not be S_ON. Suppose that a system requires N interconnection routes in the matrix, then the probability of the i-th route being successfully embedded is given by:

$$y_{ci} = (1 - y_b)^{(n + \frac{m}{2})} \left[1 - (1 - y_s^2)\right]^{(n - i)} \tag{3-27}$$

where m is the number of columns in the matrix, and n is the number of rows in the matrix (for a matrix without redundancy, n is the same as N). Further, let y_s be the yield of a single switch cell, and y_b the yield of a switch cell which is permanently shorted between its input and output.

The yield of the whole switch matrix without any redundancy is:

$$\begin{aligned} Y_o &= \prod_{i=0}^{n-1} y_{ci} \\ &= \prod_{i=0}^{n-1} (1 - y_b)^{a} \left[1 - (1 - y_s^2)\right]^{(n - i)} \end{aligned} \tag{3-28}$$

where, $a = n + \frac{m}{2}$.

Let us consider the case of the matrix with redundancy. Due to the flexible structure of the duplex matrix, there are several ways to introduce redundancy into the matrix: by adding extra columns, or extra rows, or both. However, introducing redundant rows into the matrix will not be considered further here. There are three reasons for this: firstly, IC manufacturing experience suggests that major yield loss is due to the

process of metallisation. Data collected from a processing line [20] shows that the probability of having missing or extra patterns in a metal track is much higher than for other types of defect. The matrix row failures can be mainly attributed to failures of horizontal metal tracks carrying signals from the row decoder and processing elements. However, the spacing between the horizontal metal tracks in the switch cell design is not at the critical separation. The width of these tracks is reasonably large so that the probability of obtaining such big defects (according to inverse-cube defect size distribution) and, therefore, failure of the horizontal metal track is generally low. Secondly, the area occupied by a row is twice as large as the area occupied by a column. Therefore, adding extra columns for redundancy is more economic than adding extra rows. As far as tolerating a defective switch cell within the switch matrix is concerned, the number of multiple paths between any input and output is sufficient for choosing an alternative physical route (mainly an alternative matrix row). As seen later in the yield model, this is already effective enough to tolerate defective switch cells. Adding a redundant matrix row only gives more alternatives and does not greatly improve the defect-tolerance of the switch matrix. Instead, it introduces an unnecessary amount of hardware overhead. Finally, and most importantly, by adding redundant rows into the matrix, only faults (catastrophic defects) inside the matrix can possibly be tolerated. This amount of additional complexity does not give any improvement towards tolerating any catastrophic defect inside the processing elements. Tolerating defective processing elements, however, is our main task. Therefore, adding redundant rows in the matrix is not cost-effective. A very large matrix may only need one or two redundant rows, which are mainly required to replace the matrix row affected by a defective output (SA1 or SA0) of the row decoder, or the classical logic fault of a given switch cell.

The probability of every route being successfully embedded arises from two considerations: firstly, the probability of successfully embedding a route within the non-redundant matrix area y_{ci} (the area excluding all the redundant columns); and, secondly, the probability of successfully embedding the route by using redundant columns, y'_{ci} (in the case that the routing attempt in the non-redundant area has failed because of some defective switch cells in that area). The second part should be the intersection of the event of failure to embed the route within the non-redundant matrix area and the event of successfully embedding the route in the redundant columns. The combination of these two parts forms the overall yield for embedding a route within the matrix with redundant columns. Clearly, this overall yield can not be greater than

100%. Suppose R redundant columns are put into the matrix, then the probability of the i-th route being successfully embedded can be expressed as

$$y_i = y_{ci}+(1-y_{ci})\left[\left(\prod_{j=1}^{i-1} y'_{cj}\right)(1-(1-y_s(1-(1-y_s)^{(R-i+1)}))^{(n-i+1)}\right.$$

$$+\left(\prod_{j=1}^{i-1}(1-y'_{cj})\right)(1-(1-y_s(1-(1-y_s)^{R}))^{(n-i+1)})$$

$$\left.+\sum_{j=1}^{i-2} Q_j(1-(1-y_s(1-(1-y_s)^{(R-i+j+1)}))^{(n-i+1)})\right] \qquad (3\text{-}29)$$

where, when $i \le 2$, $Q_j = 0$; when $3 \le i \le R$, the coefficient Q_j is

$$Q_j = {}^{0}C_{j}\left[\sum_{p=1}^{N_0}\left(\prod_{k=1}^{j} x_{pk}\right)\right] - {}^{1}C_{j+1}\left[\sum_{p=1}^{N_1}\left(\prod_{k=1}^{j+1} x_{pk}\right)\right]$$

$$+ \cdots + (-1)^{(i-j-1)}\,{}^{i-j-1}C_{i-1}\left[\sum_{p=1}^{N_{i-j-1}}\left(\prod_{k=1}^{i-1} x_{pk}\right)\right] \qquad (3\text{-}30)$$

where $N_r = {}^{j+r}C_{i-1}$ and x_{pk} is the k-th element of the p-th selection of j elements for the first term, of (j+1) elements for the second term, ..., and of (i−1) elements for the last term, from a total of (i−1) elements y'_{cq} (q = 1, 2, ... i−1).

In the case when i > R, the probability of the i-th route being successfully embedded becomes

$$y_i = y_{ci} + (1-y_{ci})\left[\left(\prod_{j=1}^{i-1}(1-y'_{cj})\right)(1-(1-y_s(1-(1-y_s)^{R}))^{(n-i+1)}\right.$$

$$\left.+\sum_{j=1}^{R-1} Q_j(1-(1-y_s(1-(1-y_s)^{j}))^{(n-i+1)})\right] \qquad (3\text{-}31)$$

where, when $R \leq 2$, $Q_j = 0$; otherwise,

$$Q_j = {}_{j}^{0}C \left[\sum_{p=1}^{N_0} \left(\prod_{k=1}^{j} x_{pk} \right) \right] - {}_{j+1}^{1}C \left[\sum_{p=1}^{N_1} \left(\prod_{k=1}^{j+1} x_{pk} \right) \right]$$

$$+ \cdots + (-1)^{(R-j)} {}_{R}^{R-j}C \left[\sum_{p=1}^{N_{R-j}} \left(\prod_{k=1}^{R} x_{pk} \right) \right] \qquad (3\text{-}32)$$

where $N_r = {}_{l-1}^{j+r}C$, and x_{pk} has the same meaning as that in equation (3-30). The yield of the entire switch matrix is given by

$$Y_R = \prod_{i=1}^{n} y_i \qquad (3\text{-}33)$$

Figure 3.13 shows the relationship between yield and the size of the matrix with various on-chip redundancies. The relationship between the yield and matrix size without any redundancy, but under two different defect densities, is illustrated in Figure 3.14. For two matrices with different sizes, Figures 3.15 and 3.16 demonstrate the relationship between yield and the amount of redundancy for various defect densities.

From Figures 3.13 to 3.16, several observations for the duplex matrix can be drawn:

- The yield decreases with an increase in matrix size, but not as rapidly as the normal matrix does. In fact, the yield of the duplex matrix decreases much more slowly. This is because, in the duplex matrix, a larger matrix size will offer more choices for embedding a certain connection, due to the feature of multiple paths, while in the normal matrix the number of choices for embedding a certain connection is kept unchanged with the increase of the matrix size. In other words, the probability of successfully embedding the connection is unchanged, so that the probability of successfully embedding a larger amount of connections will decrease exponentially.

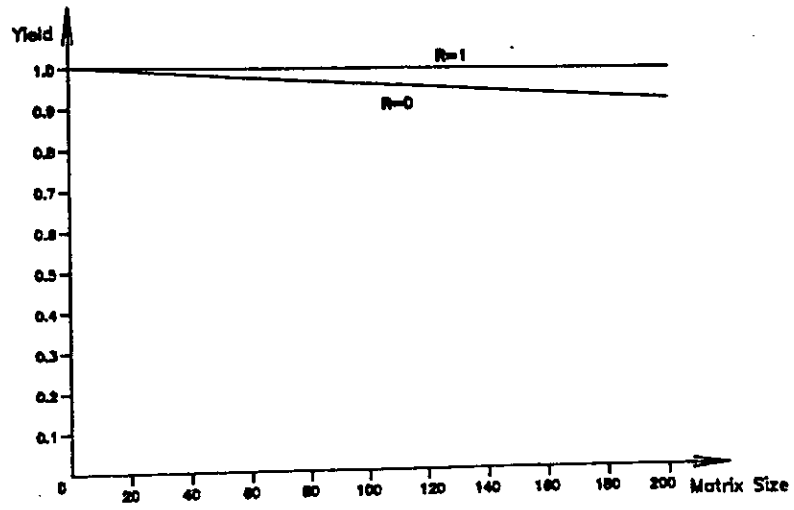

Figure 3.13 Yield vs matrix size plot with various redundancies (if the reading on X-axis is n, then the size of the duplex matrix is n×2n).

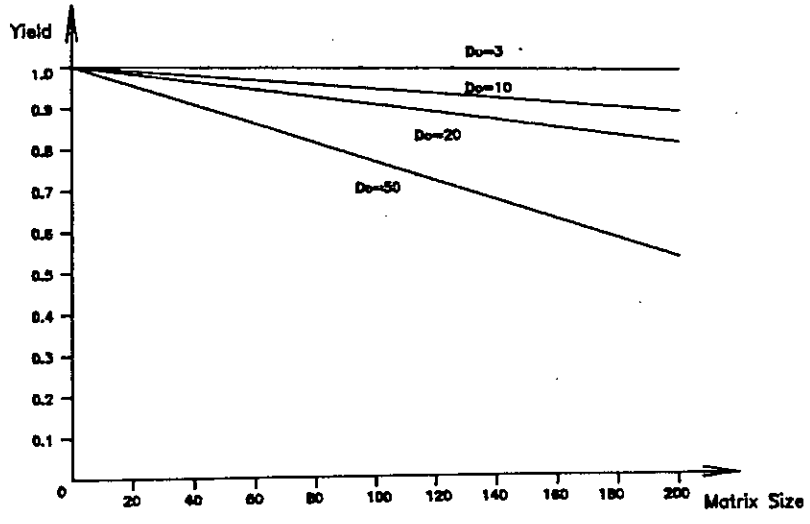

Figure 3.14 Yield vs matrix size plot with various defect densities, Do (cm^{-2}).

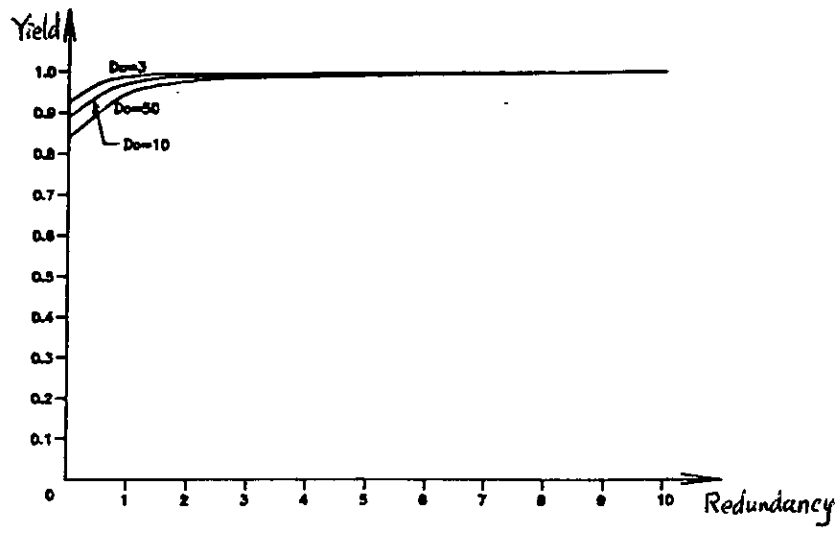

Figure 3.15 Yield vs redundancy plot with various defect densities (the matrix size is 64×128).

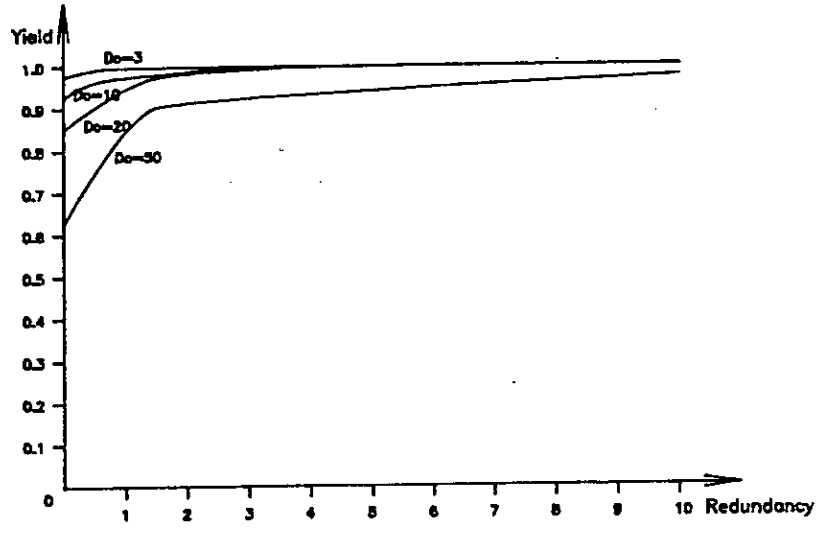

Figure 3.16 Yield vs redundancy plot with various defect densities (the matrix size is 160×320).

- The yield curves saturate after a certain amount of redundancy. In comparison with the normal matrix, the duplex matrix needs less redundancy to cause saturation of the yield curves (i.e. to get the maximum yield) and thus the duplex matrix needs less redundancy than the normal matrix.

- The increase in yield is greatest amongst the chips with lowest initial yield (without redundancy). Therefore, the low yielding chip can obtain more benefits from redundant design.

- To achieve the same yield, a larger matrix needs more redundancy, but, as compared to the normal matrix, it needs less redundancy.

Cost of Redundancy

The figure-of-merit definition for the duplex matrix, FM_d, is the same as that for the normal matrix, and is written as:

$$FM_d = \frac{A_o}{A_R} \times \frac{Y_R}{Y_o} \tag{3-34}$$

FM_d versus the amount of redundancy for various defect densities is given in Figure 3.17. Figure 3.18 presents the same relationship but for different matrix sizes. These curves show that the maximum figure-of-merit that the duplex matrix can achieve is lower than that for the normal matrix. This is because the yield of the duplex matrix without redundancy is sufficiently high that by adding redundancy the yield will not improve greatly. Instead, the increase in silicon area due to redundancy will become the dominant factor in the figure-of-merit value.

3.3.5 Other Considerations for Switch Matrix Yield

The yield models for both normal and duplex matrices developed so far are on a single route basis, as all the embedded routes within the matrix are independent of each other. Defects in the matrix can result in a change in the physical implementation of a given route, without considering the physical implementation of any other routes in the matrix. This is not generally true in a real situation in which the processing elements usually have several input and output ports connected to the matrix, so that the physical implementation of the routes relating to these input/output ports of the processing

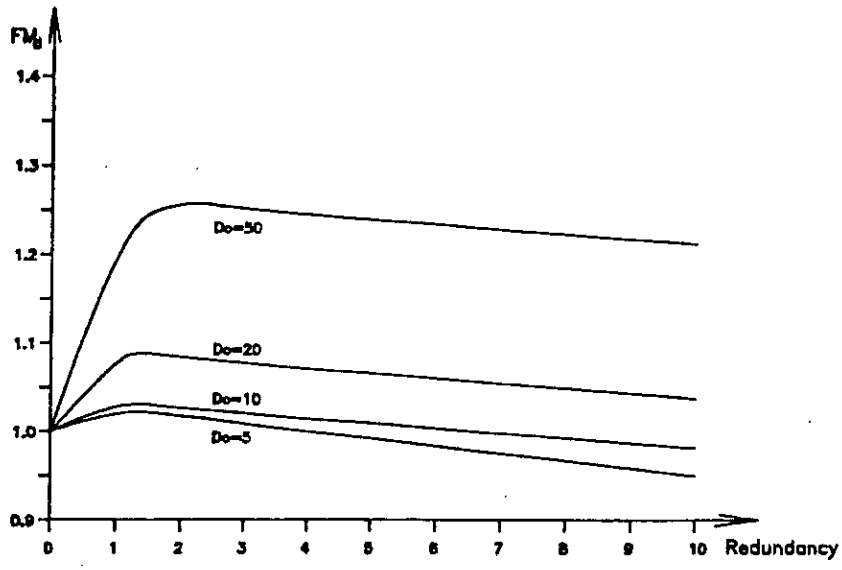

Figure 3.17 FM_d vs redundancy plot with various defect densities (the matrix size is 100×200).

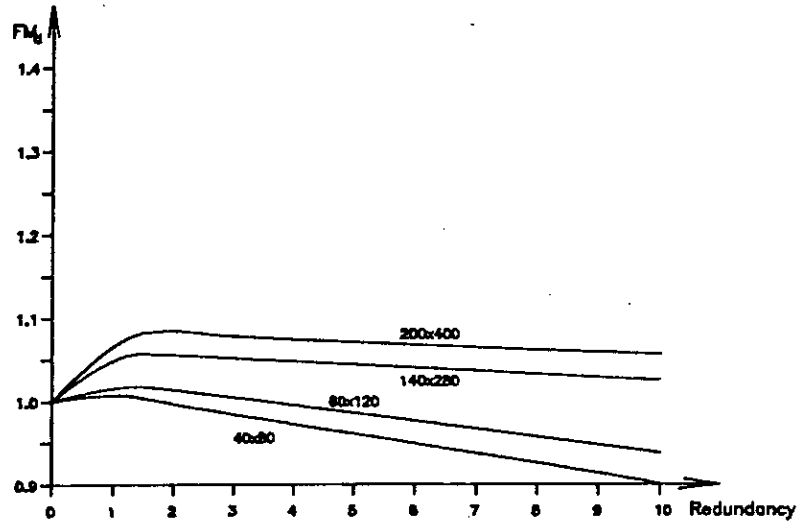

Figure 3.18 FM_d vs redundancy plot with various matrix sizes (defect density is 10 cm^{-2}).

element are likely to be dependent upon each other. For this case, any change in physical implementation (for example, using a redundant column and therefore redundant processing element) of one of these routes should also consider the changes in physical implementation of its related routes. Therefore, in general, switch matrix yield modelling should be divided into three stages, instead of two:

1. Calculate the yield value for a single switch cell and the yield value for successfully embedding a route in the matrix y_{sin}.

2. According to the number of input and output ports of processing elements, suppress the number of routes from the original number N to the number of processing elements. The new yield value for successfully embedding a route in this suppressed matrix is

$$y_{su} = \prod_{i=1}^{P} y_{si} \tag{3-35}$$

where P is the number of ports for the processing element being considered. For the normal matrix, y_{si} has the same meaning as in equation (3-22), and their values are the same for all the routes. For the duplex matrix, y_{si} is equivalent to y_{ci} in equation (3-27).

3. After suppression of the number of the routes to be embedded in the matrix, the number of columns in the matrix will shrink to the number of processing elements and be the same as the number of rows (if no redundant rows are introduced). Within this final stage, one of the yield models developed earlier for the normal matrix or the duplex matrix is applied to such a shrunk matrix using the new single route yield, y_{su}, to calculate the yield of the switch matrix.

When the yield calculation for a switch cell employs a simple Poisson yield model (as it did earlier in Section 3.3.2.), it is automatically assumed that defects are randomly distributed over the whole matrix. However, as discussed earlier, defect clustering is a common feature of a modern VLSI process. The effect of such defect clusters should be included in yield models. Furthermore, the crossbar switch matrix does not only contain an array of switch cells, as the decoding and driving circuits for the array also form part of the matrix. The yield of these circuits should also be included in the overall yield of the switch matrix. Therefore, by combining these two factors, the overall yield of a crossbar switch matrix becomes

$$Y'_R = Y_c \times Y_d \times Y_R \tag{3-36}$$

where Y_c is clustering yield. Y_d is the yield of decoding and driving circuits plus all the

other necessary auxiliary circuits†. Y_R is the yield given by equation (3-24) or (3-33).

3.3.6 Consideration of the Entire Superchip

The yield calculation for the entire superchip is based on the yield data of lower level elements such as the yield of processing elements and the switch matrix. Generalizing the problem, the processing elements contained in the superchip may not be identical. In fact, the superchip may contain m processing elements, P_a, n processing elements, P_b, p processing elements, P_c, ..., and q processing elements, P_k; each type of processing element has its own yield Y_a, Y_b, Y_c, ... and Y_k, respectively. The yield calculation of each processing element is straightforward by calculating the critical area in the processing element and by using this area in a suitable yield model in terms of the size of the processing element. Supposing the yield of the switch matrix with redundancy is Y_{SM}, then the overall yield of the entire superchip is

$$Y = Y_{sm}(Y_a)^m(Y_b)^n(Y_c)^p \cdots (Y_k)^q \tag{3-37}$$

where Y_{sm} is the yield of switch matrix without redundancy. If there are R redundancies for P_a, then the yield of this processing element with redundancy can be expressed as

$$Y_A = \sum_{i=0}^{R} C_{n+R}^{n+i} (Y_a)^{(n+i)}(1 - Y_a)^{(R-i)} \tag{3-38}$$

This expression can equally be applied to P_b, P_c, ... and P_k. Therefore, the yield for the whole superchip with redundancy can be written:

† Strictly speaking, simply regarding the yield value of the control circuits (mainly decoding circuits) as a gross yield in the equation will not be entirely correct, because one or two faulty lines in these decoding circuits may not be destructive to the whole switch matrix (particularly the column decoder). For example, we can use a redundant matrix column introduced for tolerating defective switch cells and processing elements to get around the problem. However, as seen in the prototype analysis later, any fault in the control circuits will do a great deal of damage to the switch matrix. It is hoped that by adding Y_d a reasonable amount of redundancy for the possible faults in the control circuits can be achieved. For instance, the number of redundant rows should depend on the yield of row decoding circuits, and some extra columns and processing elements are needed to tolerate faults in column decoding circuits.

$$Y = Y_{SM} Y_A Y_B Y_C \cdots Y_K \tag{3-39}$$

According to the earlier analysis, Y_{SM} can be reasonably high (more than 80%) even if the size of the matrix is large. In order to get a reasonable yield, Y_A, Y_B ... and Y_K should again be reasonably high. This requires a high yield for each processing element. For example, if there are 10 processing elements, P_a, in the chip and each of them has yield of 50%, then the overall yield Y_A for P_a will never reach 3% no matter how many redundant elements are included in the chip. Hence, the overall yield of the chip will never reach 3% either.

3.3.7 Redundancy Selection

When the yield calculated by equation (3-37) can not meet the required yield Y_{req}, a necessary amount of redundancy should be added into the chip to improve the yield. The algorithm for selecting and calculating the required amount of redundancy is shown below, in expanded form:

1. Calculate each group of elements Y_{sm}, Y_a, Y_b, Y_c, ... Y_k.

2. Calculate the yield of the system without redundancy Y as shown in equation (3-37).

3. If the calculated yield value satisfies the specified yield value Y_{req} ($Y \geq Y_{req}$), calculation terminates successfully. Otherwise, set a *yield deviation* defined as the difference between the current yield value minus the previous yield value,

$$DY = Y_{curr} - Y_{prev} \tag{3-40}$$

and carry on to step 4.

4. Choose the processing element with the lowest yield among Y_A, Y_B, Y_C... and Y_K, using Y_a, Y_b, Y_c, ... and Y_k, as a first target. Add one such redundant element to this group, and update the yield value of this group within Y_A, Y_B, Y_C... and Y_K. Increment the switch matrix row and/or column (according to the type of matrix) by the number of I/O ports required to connect an extra processing element into the system. Then, update the yield, Y_{SM}, with the introduced

redundancy. Note that Y_{SM} should be calculated in three stages indicated in Section 3.3.5.

5. Use equation (3-39) to update the yield. Compare this yield Y to Y_{req}, and if $Y \geq Y_{req}$, the calculation terminates successfully. Otherwise, update the "DY". If DY is greater than a deviation limit set by designer, proceed to step 4, otherwise the calculation terminates unsuccessfully†.

Optimal redundancy is derived from the evaluation of the figure of merit. The peak value of each curve in Figures 3.11, 3.12 and 3.17, 3.18, for example, indicates the amount of redundancy which results in the most economic hardware overhead to obtain the greatest possible improvement in yield. From these curves, it is clear that the optimal amount of redundancy increases with an increase in matrix size.

3.4. Optimal System Block Partitioning Using Crossbar Networks

For a given system architecture using the crossbar switch matrix, there can be several physical configurations. For instance, a complex multiplier can be regarded as a processing element, or it can be sub-divided into six smaller processing elements, four real multipliers and 2 adders, linked by the switch matrix. These two different configurations result in different complexities of the processing elements, and require a different size for the crossbar switch matrix. A question arises here, which configuration is better in terms of the cost-effectiveness of defect/fault tolerance? Generally speaking, for a particular system (assuming that the complexity at a system level is constant), the question may be recasted as; what is the optimal block size of the processing element in the superchip? This topic is addressed in this section. The partitioning range for the processing elements (a range of sizes), within which the cost-effectiveness of fault-tolerance through the crossbar switch matrix can be achieved, will be discussed here.

Let us take the two extremes. If a complex system comprises processing elements at a very low level (NOR, NAND gate level), then to implement such a system in superchip style, a huge crossbar switch matrix is needed to connect all these small

† Unsuccessful termination means that the required yield is too high to be met by a given manufacturing technology, even after a large amount of redundancy is included in the chip.

processing elements together. On the other hand, if the processing elements are of a high complexity (ALU, multiplier, etc), a small switch matrix is needed to realise the whole system. In both cases, the complexity of total functional circuits (excluding switch matrix) is constant. In the former situation, the yield of each individual processing element will be very high. Here, the defect-tolerance of the processing elements may be effective in the sense that the sacrifice for any critical defect is very small (probably one small logic gate). However, the yield of the extensive crossbar matrix may be so poor that it will dominate the yield of the whole system. In fact, the system yield may have been even lower than that of a non-programmable case, e.g. such as a FIRST implementation discussed in Chapter 2. Also in this case, a large proportion of the silicon area is used by the switch matrix for system communication instead of computation. This is undesirable from an economic point of view. Therefore, there is little benefit in using the crossbar switch matrix in this extreme case.

In the latter situation, adding a large amount of redundancy is not economic, because the processing elements are so big that appending a single one into the superchip results in a significant hardware overhead. Therefore, the level of redundancy will be kept very low resulting a very limited useful defect-tolerant capability. In other words, when there is any critical defect in the system, the system implemented in superchip style does not have many alternatives (redundancies) to choose to avoid the defective processing elements. Ultimately, the defect-tolerant capability in this extreme case is quite similar to its non-programmable counterpart.

To measure the cost-effectiveness of using a crossbar switch matrix in a superchip, a new figure-of-merit parameter, FM_p, is defined here for the whole system:

$$FM_p = \frac{A_N}{A_P} \times \frac{Y_N}{Y_P} \tag{3-41}$$

in which

A_N: area occupied by the system without the crossbar switch matrix (non-programmable case).

A_P: area occupied by system using the crossbar switch matrix (programmable case).

Y_N: yield of system without using the crossbar switch matrix (by fixed wiring).

Y_P: yield of system using the crossbar switch matrix, including the yield of the switch matrix and the yield of the processing elements.

Given the complexity of a system, different partitioning† results in a different complexity for each individual processing element. For each partitioning, a different degree of redundancy will result in a different value for the figure of merit. However, the figure of merit for a given partitioning has its highest value at a particular amount of redundancy. This maximum figure of merit value represents the best possible return using partitioning. If this value is less than unity, it means that there is no benefit in using the switch matrix for that partitioning. Figure 3.19 shows the highest value of FM_p that a particular partitioning can achieve versus various partitions shown by the matrix size, in which the normal matrix architecture is used. The curves in the diagram represent two different overall system complexities as indicated in the diagram. They show

- for the same crossbar switch matrix size, more complex (larger in required silicon area) processing elements will obtain more benefit from using such a switch matrix.
- more complex systems allow a wider range of partitioning within which the switch matrix will bring benefit for the yield improvement ($FM_p > 1$).
- a specific partitioning of a system which results in a large population of very simple processing elements will lead to an unacceptable system cost, indicated by the fact that the obtainable highest value of FM_p is below unity.

Figure 3.20 illustrates the relationship between the system complexity and the beneficial partitioning range for the normal matrix. This partitioning range is represented by the maximum number of matrix rows and columns (the maximum size of the switch matrix) beneficially partitionable for a particular system complexity (required silicon size).

† In the rest of the discussion, we assume that a system is partitioned into a number of processing elements each of which has the same complexity.

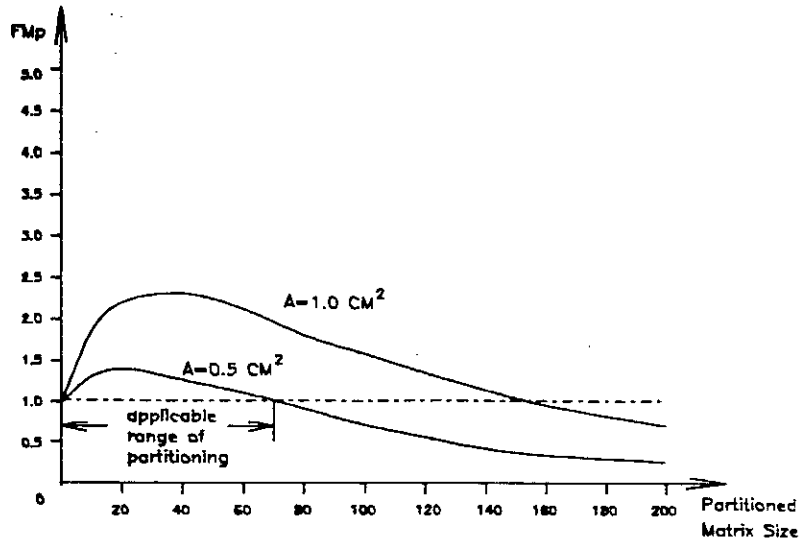

Figure 3.19 FM_p vs different partitions for a particular system complexity in the normal matrix organisation (Here the critical area is supposed to be 10% of the total silicon area, and A is the total silicon area occupied by all the processing elements).

Similarly, Figure 3.21 and 3.22 are the two equivalent diagrams to Figure 3.19 and 3.20 but for the duplex matrix. Again, the partitioning range is indicated by the maximum number of matrix columns for a particular system complexity. In Figure 3.22, it can be seen that, with the increase in system complexity, the beneficial partitioning range goes up very rapidly. This results mainly from the fact that the duplex matrix has inherently a much higher yield than the normal matrix. This applies particularly when the matrix size is very large, so that even for very fine partitioning (partitioning with low complexity of processing elements) the duplex matrix still functions correctly as a defect-tolerant mechanism rather than giving the system an additional yield burden (as the normal matrix does at the same situation). It can also be seen from Figures 3.19 and 3.21 that the partitioning resulting in the maximum value of FM_p brings the block size of the processing elements to around 5 mm^2. With current 2 or 3μm, double-metal, CMOS technology, a transistor count for the processing elements between 10^4 to 10^5 seems to be the optimum block size for the superchip architecture.

3.5. Block Partitioning versus Hierarchical Redundancy

All the discussion and measures developed earlier are based on a single level redundancy scheme, in which redundancy is only available at a processing element level. However, today's VLSI design sequence is usually based on a structural hierarchical design methodology in which a chip is partitioned into different levels of

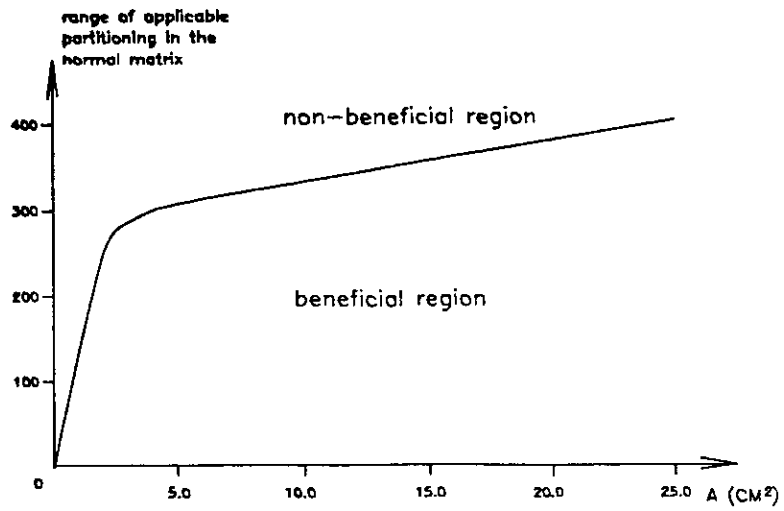

Figure 3.20 Range of applicable partitioning vs system complexity for the normal matrix (suppose Do is 10 cm^{-2}, total critical area occupies 10% of the total silicon area).

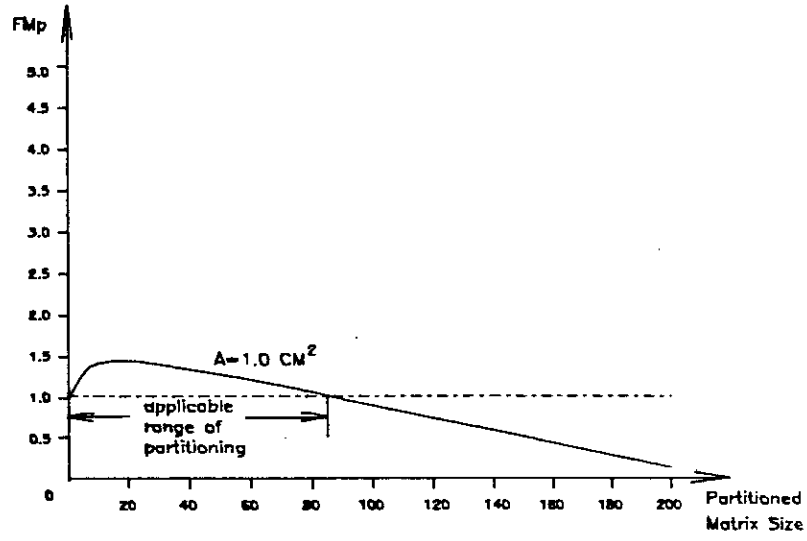

Figure 3.21 FM_p vs different partitions for a particular system complexity in the duplex matrix organisation (Here the critical area is supposed to be 10% of the total silicon area, and A is the total silicon area occupied by all the processing elements).

building blocks, according to their functional relationship. In this case, optimal partitioning may not be achievable, nor may a good (beneficial) partition be guaranteed. More importantly, the size of processing elements obtained from an optimal partition, as presented in the last Section, may not meet the high yield requirement for each individual processing element, mentioned at the end of Section 3.3.6. Low yield for the processing elements can result in a very low, and unacceptable, overall yield for the superchip, although it can be much better than its non-programmable counterpart. It follows

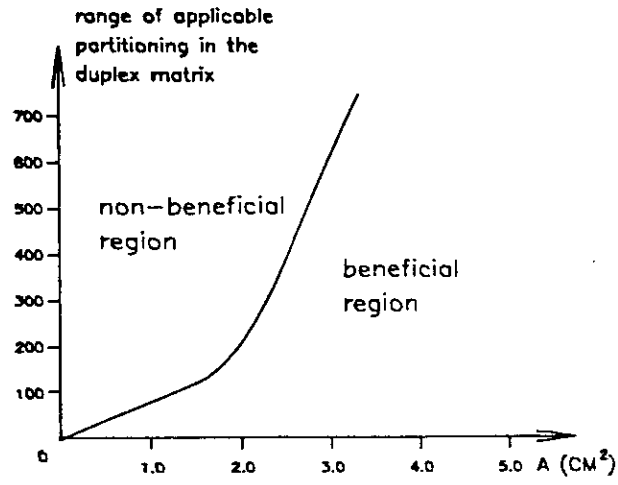

Figure 3.22 Range of applicable partitioning vs system complexity for the duplex matrix (suppose Do is 10 cm^{-2}, total critical area occupies 10% of the total silicon area).

from these considerations that we should think how hierarchical redundancy may ease these conflicts. Huang and Daughton [118] studied yield and chip cost for the case where two levels of redundancy are taken into account. They assumed that a VLSI chip consists of N_m modules, each module containing N_e elements. If the overall hardware overhead for redundancy is fixed, then different distributions of redundancy at two levels may result in different yield values, and therefore cost†. They concluded that when the number of elements per module is much larger than the number of modules per chip, it is not beneficial to use two levels of redundancy. Instead, redundancy should mostly be employed at the lower level (the element level). When the number of elements per module is comparable to, or smaller than, the number of modules per chip, then it is beneficial to use two levels of redundancy.

The superchip architecture offers two levels of redundancy by introducing a small switch matrix into the processing elements, as well as a larger matrix being used for processing element interconnection. As indicated in Section 3.3.6, however, when processing elements are not large, the best yield can be achieved with only one level of redundancy at the processing element level. When processing elements are large and complex (this implies fewer processing elements if the system complexity remains

† A single level of redundancy is the special case here when the distribution at one level is zero.

constant), two levels of redundancy may be useful to enhance the yield of an individual processing element as well as the yield of the entire superchip. Taking the example given at the end of Section 3.3.6 again, if we assume that a certain amount of redundancy is introduced at a lower level inside P_a so that its yield rises from 50% to 80%. Thus the yield for 10 processing elements, P_a, can reach 73% through the higher level of redundancy (the redundancy at the P_a level). Clearly this is a considerable yield enhancement achieved with two levels of redundancy, in comparison with the previous 3% value achieved with a single level redundancy. This comparison will be exemplified in more detail in the case study contained in Chapter 6.

3.6. Conclusions

At the beginning of this Chapter, various yield models for predicting the processing yield of integrated circuits were reviewed. Some of these were established in 1960s and have had to be modified in line with VLSI manufacturing technology trends. Thus more complicated analytical, but also empirically-based, yield models have been proposed to track the technology. A summary of these yield models has already been presented at the end of the Section 3.2.10. This was followed by the analysis of the fault mechanism within the switch cells, which are the most basic building blocks of the switch matrix. This was the starting point of the numerical analysis of the switch matrix, leading towards the establishment of a yield model for each switch matrix. The analysis shows that various processing defects within the switch cells can result in the following functional fault:

- *Switch Cell S_ON*: the input and output of a switch cell are permanently connected so that the on-off state can not be controlled externally. This fault can possibly be tolerated within the matrix without affecting the processing elements.

- *Switch Cell S_OP*: the input and output of a switch cell are permanently disconnected so that the on-off state can not be controlled externally. This fault can possibly be tolerated within the matrix without affecting the processing elements.

- *Switch Cell SA0 or SA1*: the output of a switch cell is permanently stuck at 1 or 0 so that any signal through the switch is likely to be corrupted. This kind of fault can not be tolerated within the switch matrix itself.

From a fault-tolerant point of view, more emphasis has been placed on developing tolerant techniques for the first two faults alone. Any SA0 or SA1 faults within a switch cell will cause the whole related matrix column or row to be SA0 or SA1. This simply results in the abandonment of the related matrix column or row.

A functional yield model was developed for each switch matrix, based on individual yield values for the switch cells. The yield values were derived from one of the statistical yield models previously reviewed. This involved the calculation of the critical geometry within the actual switch cell layout. Figures of merit were also defined. The results prove that deliberately introduced architectural redundancy is very effective in yield improvement under certain conditions. The optimal amount of redundancy can be obtained from the figure of merit estimation for a system. An algorithm for calculating the yield figure for the whole superchip, and selecting the necessary amount of redundancy for achieving a desired yield value, was given in Sections 3.3.6 and 3.3.7.

System partitioning is another important issue. Consideration was given to finding a cost-effective way of dividing a system into suitable sized processing elements implemented in the superchip style. The results show that the effective partitioning range goes up with increased system complexity. The growth rate depends upon the structure of the switch matrix. The best transistor count for the processing elements in the superchip seems to be between 10^4 to 10^5. Coincidentally, the node complexity of several large processor array chips existing or currently under development [119] also lies within or near the bounds of this range. However, when a partition (within the partitioning range) makes the yield of individual processing elements very low, a second (lower) level of redundancy should then be introduced, rather than employing a large amount of redundancy at the processing element level, which may still not enhance the yield up to a reasonable level. By adjusting degree of redundancy at both levels, a considerable amount of yield improvement can be achieved. An example given in Section 3.5 showed that such hierarchical redundancy may enhance the yield much more effectively than single level redundancy.

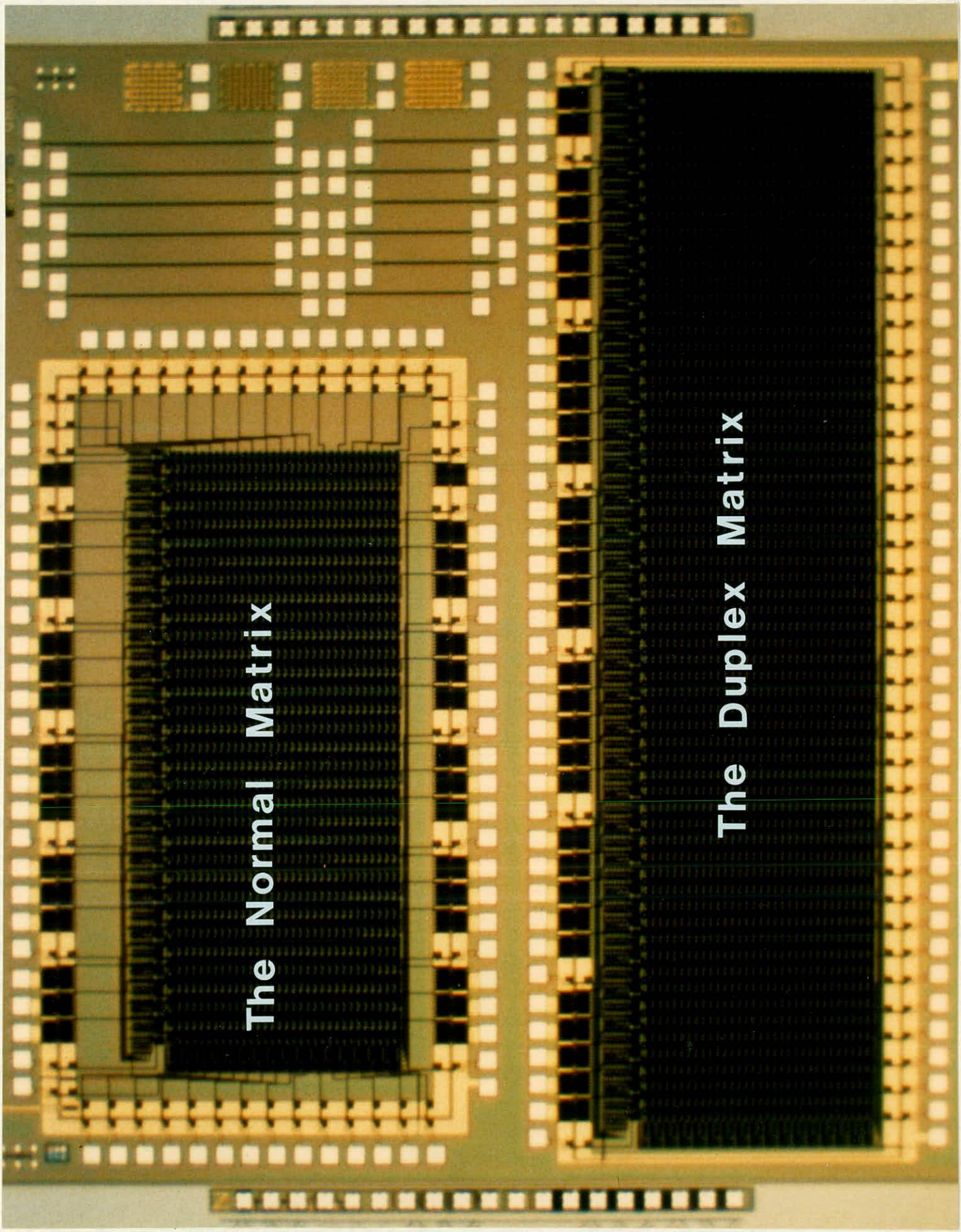

Plate 1, Microphotograph of the two switch matrix prototypes fabricated by Plessey Co.

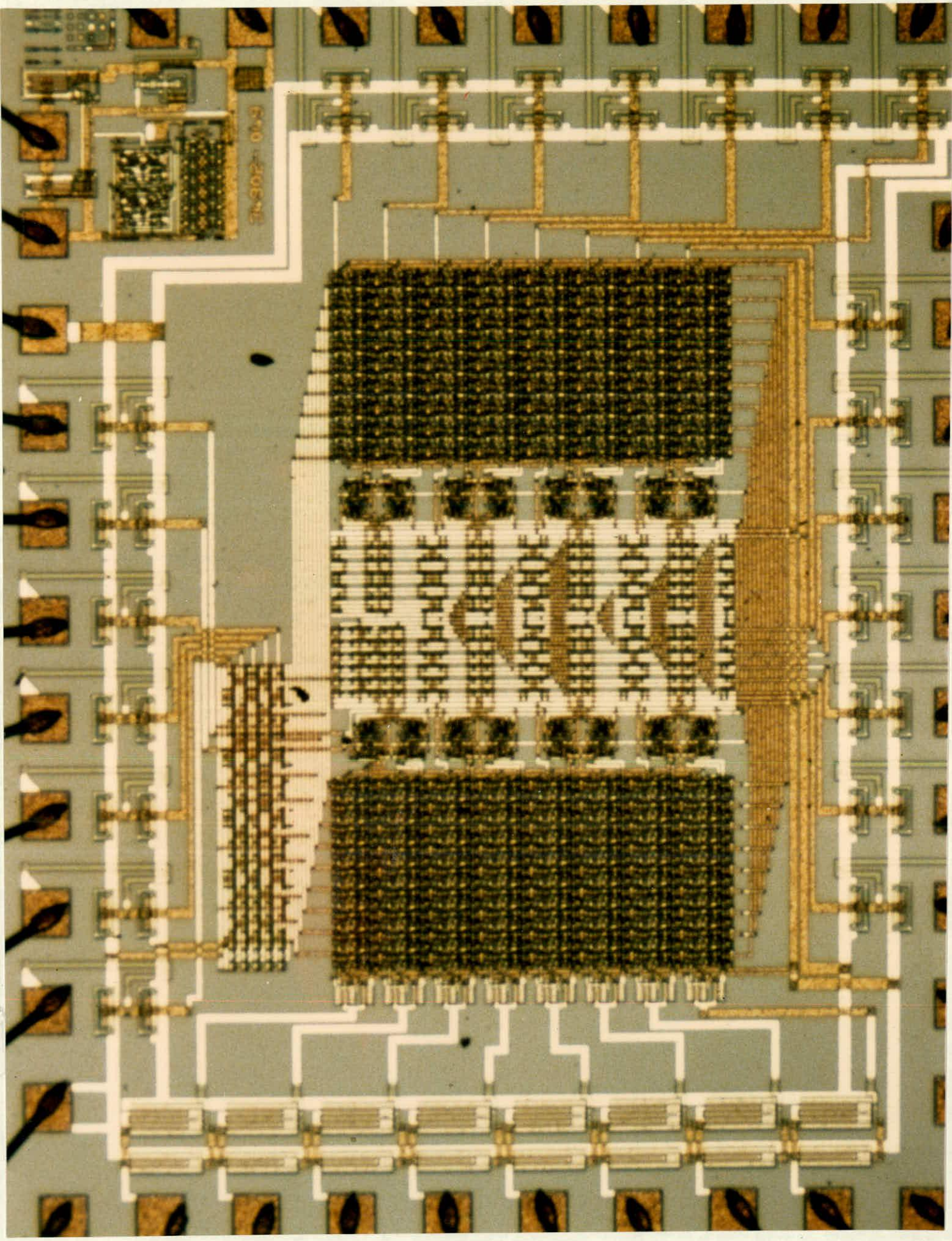

Plate 2, Microphotograph of the switch matrix prototype fabricated by MCE Ltd.

CHAPTER 4

THE PROTOTYPE CROSSBAR SWITCH MATRIX DESIGN AND EXPERIMENTAL RESULTS

4.1. Introduction

The concept and organisation of the superchip were introduced in Chapter 2. An improvement in defect/fault tolerance and system reconfigurability were indicated by using the crossbar switch matrix, in comparison to its non-programmable version (e.g. a FIRST-compiled chip). Although extensive simulations of the switch matrix design have been undertaken, it is still important to validate the design in silicon. Furthermore, certain performance parameters of the crossbar switch matrix e.g. signal delay, distortion etc, play an important role in the overall system performance. Three prototype crossbar switch matrices were designed using 2.5μm and 3μm, double-metal, CMOS technology available at the Plessey Co. and MCE Ltd, to evaluate its functionality and characteristics. Plate 1 shows two prototype designs fabricated at Plessey Co. One is a 32 by 32 switch matrix organised in the normal switch matrix form shown in Figure 2.6, the other is a 32 by 64 switch matrix organised in the duplex switch matrix form shown in Figure 2.8. Plate 2 illustrates another design fabricated at MCE Ltd, which is 9 by 16 switch matrix organised in the duplex matrix form. This Chapter presents the test results from these prototypes and compares them to the simulation results.

4.2. Prototype Design Procedure

Each prototype chip consists of a column decoder, a row decoder, an array of the switch cells, some NAND gates, and "Tesc" circuit pairs (described earlier in Chapter 2). All these circuits and their organisation are presented in Figure 2.13. For the duplex matrix, a signal path from an input pad through two switch cells to an output

pad brings a situation in which the second switch cell has to drive a very large capacitive load (a large buffer associated with an output pad). This will cause a serious delay and a degradation of the performance of the switch matrix. In the prototype designs, therefore, each output from a matrix column passes through an intermediate buffer which is designed to avoid the above situation, Suppose that the buffer in output pad is larger than the intermediate buffer by a factor f, the same as the intermediate buffer and the switch cell. It has been shown [120] that value of f ranging from 2 to 4 are optimal in terms of time delay. In the prototype, f was chosen to be 3.7, both for the switch cell to the intermediate buffer and for the intermediate buffer to the output pad.

For convenience in further discussion, let us name all the input and output pins available for testing as IN0, IN1, IN2, ..., and output pins as OUT0, OUT1 OUT2, ..., respectively. The addressing signal input has 5 or 6 bits for the matrix row decoder and the matrix column decoder depending upon different sizes of the matrix; these are labelled, from the least significant bit to the most significant bit, as RDEC1 to RDEC6 for the row decoder, and CDEC1 to CDEC6 for the column decoder. The "disabled state" control pin, "Dset", and the "clearing state" control pin, "Tset", (discussed in Chapter 2, Section 2.4.2) are named here as ANDIN and TESTIN.

The process of layout and its verification were carried out with a suite of CAD tools. Most work was based on tools which originated from the University of California, Berkeley, and were then modified and released by the University of Washington and the VLSI Consortium [121,122] in the USA. These include CAESAR (an interactive graphics editor), LYRA (a design rule checker), MEXTRA (a circuit extractor), RNL (a switch level simulator), and SPICE (a circuit level simulator).

4.2.1. Layout Generation

Full custom chip design is very time consuming and prone to human error during layout. The approach adopted in this research for generating the whole layout for the prototypes was a cell based approach. All the circuits used in the chip, such as switch cell, NAND gate, inverter, and "Tesc" circuit, etc., were laid out individually using the graphics editor CAESAR. The three custom layouts of both swc-1 and swc-2 have already been illustrated in Figures 2.10, 2.11, and 2.12. Some 'tiling' blocks for generating row and column decoders were also designed. The layout data of these cells are stored in a library, which also includes input and output pads. A layout assembler was

designed by the author during the development of the prototype and written in C language [123] and CLAP (a C-based layout language). [124] According to the matrix size specification and its organisation (normal matrix or duplex matrix) presented to the assembler, the latter can generate automatically all the necessary building blocks and connect them together to form the desired switch matrix chip. The assembler defines and assembles new large cells, such as the row and column decoders, from the library before connecting them up to the whole chip. The assembler output is in CIF (Caltech Intermediate Form) format [120]. Prototype design layoutsof the switch matrix shown in Plates 1 and 2 were generated by the assembler.

4.2.2. Design Verification

Design verification was performed hierarchically. Firstly, verification was carried out within the cell library. Basically, each cell in the library was extracted using MEXTRA, to switch level and circuit level. Thereafter, switch-level simulation was undertaken using the RNL switch-level simulator, and, additionally, the circuit level simulator SPICE confirmed the functionality and the timing information for the cells. Finally, the same circuit extraction routing, and then the switch level simulation were applied to the whole MCE chip (the two Plessey chips are too complex for the tools to perform whole chip verifications). Figure 4.1 shows the result of such a switch-level simulation on the MCE prototype extracted from the actual layout, including all the input/output pads. All the input signals are exactly the same and applied simultaneously to the respective input pads, so only one input signal is shown in the diagram. (I_i,O_j) is used to indicate a signal path from the i-th input to the j-th output. All the loading signals are arranged such that the first matrix row is assigned to a path (I_1,O_1), the second matrix row is assigned to a path (I_2,O_2), ... and the eighth matrix row is assigned to a path (I_8,O_8). These paths are serially established as shown in the simulation result. As the switch matrix is organised according to the duplex matrix organisation, two switch cells on a matrix row are needed to be activated in order to establish a signal path. In the simulation, these two switch cells are activated (turned on) one after another according to the required signal path inside the matrix. Figure 4.2 illustrates the loading signals for establishing a signal path in the duplex matrix during the simulation, which, in principle, follow the loading scheme developed in Section 2.3.4 and shown in Figure 2.17. The simulation result shows that all the 8 outputs track their inputs following the appearance of the second pulse of the "Dset" signal, i.e. just after

the physical path is established for that connection. The loading scheme for the normal matrix has been shown in Figure 2.16. Only one switch cell requires to be activated to establish a signal path in the normal matrix.

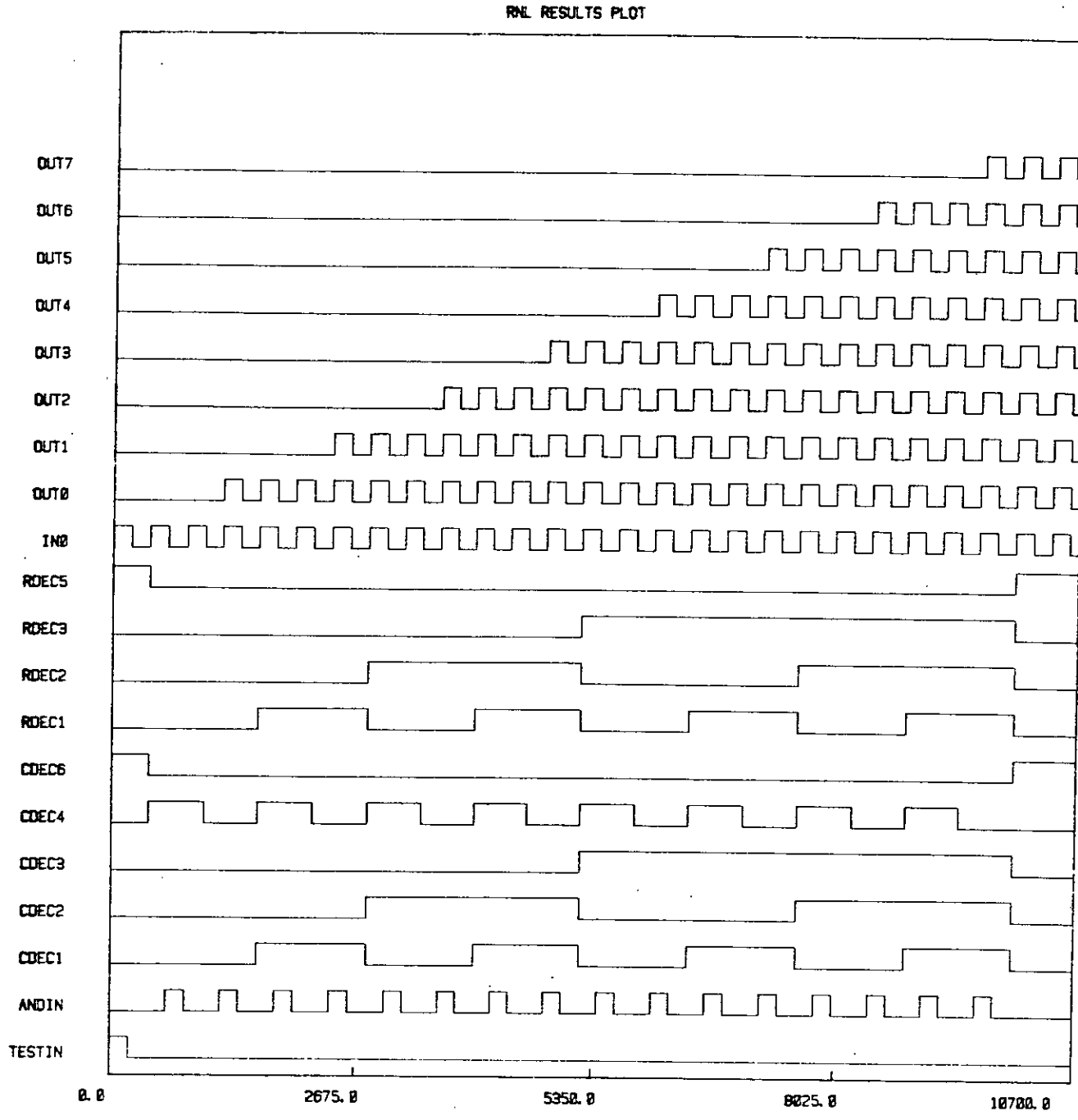

Figure 4.1 Switch level simulation of the whole MCE chip

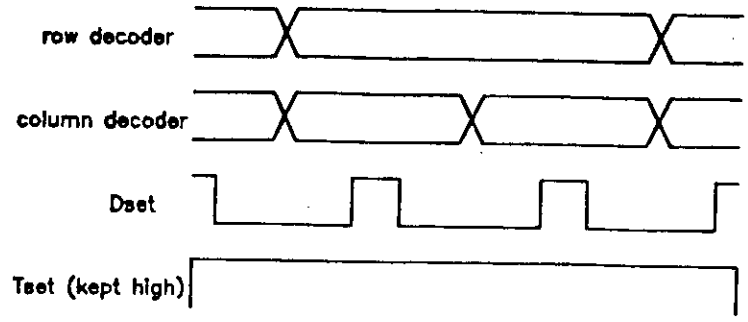

Figure 4.2 Loading signals for establishing a signal path in the duplex matrix

4.3. Test Procedure for the Prototypes

The fault pattern must be captured before a switch matrix can be used. This pattern includes the location and fault type of all the defective switch cells in the matrix, and the location of all the faults present in the decoding and driving circuits. The chip test exercise should sensitize all the switch cells and determine all the test outputs from the cells being tested. Due to the limited number of I/O pins available in the prototypes, particularly the duplex matrix chips, the approach adopted was to use the decoding and driving circuits to pass the test signals on to each switch cell in the matrix. Note that these decoding and driving circuits were initially untested at the time when they were being used to access the switch cells.

S_OP and S_ON are the two possible faults in the switch cells to be detected. The testing scheme used for these two fault conditions was very straightforward and shown in Figure 4.3 for swc-1, and Figure 4.4 for swc-2. Basically, if a switch cell is turned off and its output still follows the input, then this cell is stuck-on (S_ON). On the other hand, if it is turned on and its output does not follow the input, then this switch cell is stuck-open (S_OP). The entire switch cell array was tested on a row-by-row basis. Remembering that when "Tset" is set to logic one, all the switch cells on one matrix row may be activated by addressing that matrix row through the row decoder. Thus the testing of one matrix row can be divided into three phases:

I Initialise the whole matrix by setting both the row decoder and the column decoder to their "clearing state" (see Chapter 2). Thus all the switch cells in the matrix are intentionally turned off.

II Address a particular cell on the matrix row. This switch cell must be located on the matrix column to which the exercise signal ("input" signal in Figure 4.3) is applied. Activating this cell is achieved by passing the exercise signal on to the matrix row, so that all the other switch cells under test in that matrix row may be exercised (as Figure 4.3 illustrated). This phase is only for the duplex matrix.

III Address a matrix row and exercise all the switch cells on that matrix row. For the duplex matrix, this matrix row is the one that the switch cell activated in the second phase resides.

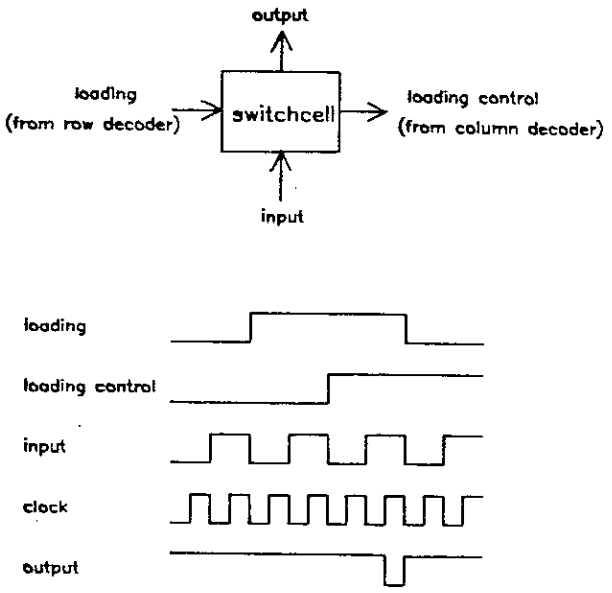

Figure 4.3 Dynamic switch cell (swc-1) testing

The choice of which particular matrix column to use in the second phase is not important. In fact, any column connected with an input pad can be chosen. During such a test, all the input pads except the one where the test signal is applied are considered as output pads for observing the response. This is possible because the structure of the input pad in the chip only consists of a protection circuit (two reversed diodes) for removing possible spikes of input signals, therefore, it can transfer signals

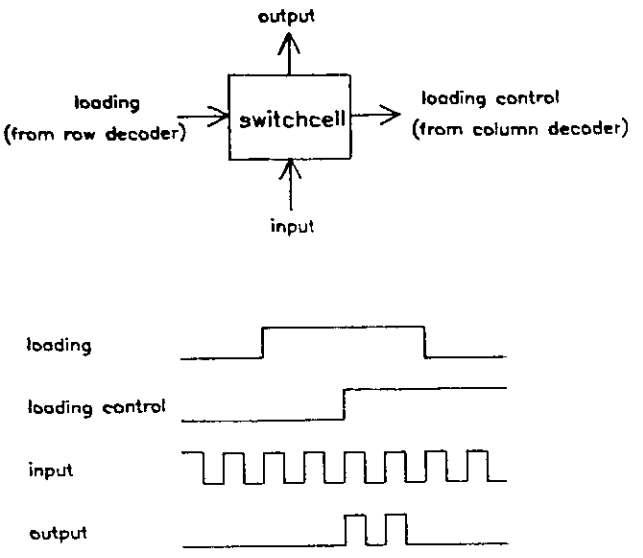

Figure 4.4 Static switch cell (swc-2) testing

bidirectionally. By observing signal response at the input and output pads connected to the matrix columns, the following information can help to distinguish the fault types and locations for the switch matrix under test:

I If the exercise signal (usually a square waveform) appears at any of the observing points during the first phase, then some short-circuits must exist between these matrix columns where the exercise signal appears and the matrix column or row where the exercise signal is applied. For a perfect switch matrix, there should not be an exercise signal appearing at any of these observing points.

II For the duplex matrix, if the exercise signal appears at any of the observing points during the second phase, then the switch cells located at the crosspoint of the matrix row which is currently being tested and the matrix columns where the exercise signal appears are stuck-on (S_ON). For a perfect switch matrix, there should not be an exercise signal appearing at any of these observing points.

III During the third phase, the test signal should start appearing at all the observing points. If any of these observing points exhibits the test signal during this phase, then the switch cell at the crosspoint of the matrix row being tested and the column where the exercise signal appears is working. For the duplex matrix, it

also indicates that the switch cell turned on during the second phase to pass the exercise signal on to the matrix row from the exercise signal input pad is not S_OP (i.e. may be working, or S_ON). This switch cell will be referred to as the *testing cell*. If the exercise signal does not appear at SOME outputs during the third phase when it should, then it can be concluded that the switch cells on the cross point are S_OP, and the testing cell is functioning.

IV If the test signal does not appear at all at any observing point during the third phase, several possible scenarios could apply. The first possibility is that the testing cell (for the duplex matrix case) was not actually turned on due to its S_OP fault, so that no exercise signal is passed on to the switch cell under test. The second possibility is that the intended matrix row was not properly addressed. In other words, the intended output from the row decoder may not be set to logic one (SA0) to activate the matrix row. Therefore, no information has been written into any switch cell on that row (including the testing cell). The third possibility is that all the switch cells on that row are S_OP. Statistically, there is little chance of the last situation happening, thus it is basically not considered. To check whether the second case applies, another switch cell on different matrix column can be chosen as a new testing cell, with which the test routing is repeated. If the result shows that the testing cell can be turned on and there are some outputs at the observing pads during the third phase, then it can be concluded that the row decoder is working for this matrix row. Otherwise, it is very likely that the row decoder is not working for the current matrix row.

One uncertainty about the duplex switch matrix from the above tests is that, if the testing cell is S_ON, then the test results will be exactly the same as that for a perfect working matrix. However, this can be checked by assigning another switch cell, which has been shown to be working during the test, to be the new testing cell to see if the ex-testing cell is S_ON.

Figure 4.5 illustrates the RNL switch level simulation for such a test to the MCE prototype chip (defect free). Again, the two Plessey chips are too complex to perform such simulation. The simulation only shows the results of testing two matrix rows: the first and the second row. When "RDEC5" "TESTIN" are both set to logic one, both decoders are in the "clearing state" and therefore, the whole matrix is being initialised. This corresponds to the first phase of the test. Then "RDEC5" is set to logic zero,

letting the row decoder address a testing matrix row (the first, and then the second row in the diagram). The first half of the time, with "RDEC5" set to logic zero, is used to activate the chosen testing cell on the matrix (the switch cell chosen in the simulation is the one connecting the "IN0" pin). This is related to the second phase of the test. The third phase is carried out in the second half of the time, with "RDEC5" set to logic zero, during which "TESTIN" is set to logic one. Thereby the whole matrix row is activated and the exercise signal starts appearing at the observing points.

4.4. Test Results

4.4.1. Generating Fault Patterns

The test scheme discussed earlier can test not only the faults in the switch cell array but also those in the matrix row decoder. Row decoder faults are assumed to result in an absent logic one for certain output line(s). This is equivalent to a situation where all the switch cells on that (those) matrix row(s) are S_OP. Thus any fault in the row decoder can be mapped into the switch cell array by always regarding the row decoder as perfect. For the matrix column decoder, the same mapping process can be undertaken. One extreme case may be such that there is no output at all at any matrix row. This might be caused by complete failure of the row and/or the column decoder, caused by failure of the NAND gates or the "Tesc" circuits attached to the column decoder. This situation was observed during the test of the MCE prototype.

A 9100 series Tektronix Digital Analysis System (DAS) [125], and its accompanying UNIX supporting software suite [126], was used to perform the tests. Figure 4.6 shows the output data for one MCE chip obtained in response to test signals, exactly the same as those shown in Figure 4.5. Because only eight signals at maximum can be displayed on the screen of the DAS, only RDEC5 is shown in the diagram labelled as "TEST" at the middle of the diagram to give timing information for the test signals. Signals from the seven remaining input pads and the eight output pads are illustrated above and below the "TEST" signal, respectively. Nine matrix rows were tested in series. The result shows that there are two permanent shorts between "IN0" (where the cyclic square signal is applied) and "IN5", and also between "IN0" and "OUT7". From a user's point of view, this is equivalent to the situation where three switch cells are S_ON, each of which is on one of these three columns, and all of which are on the same

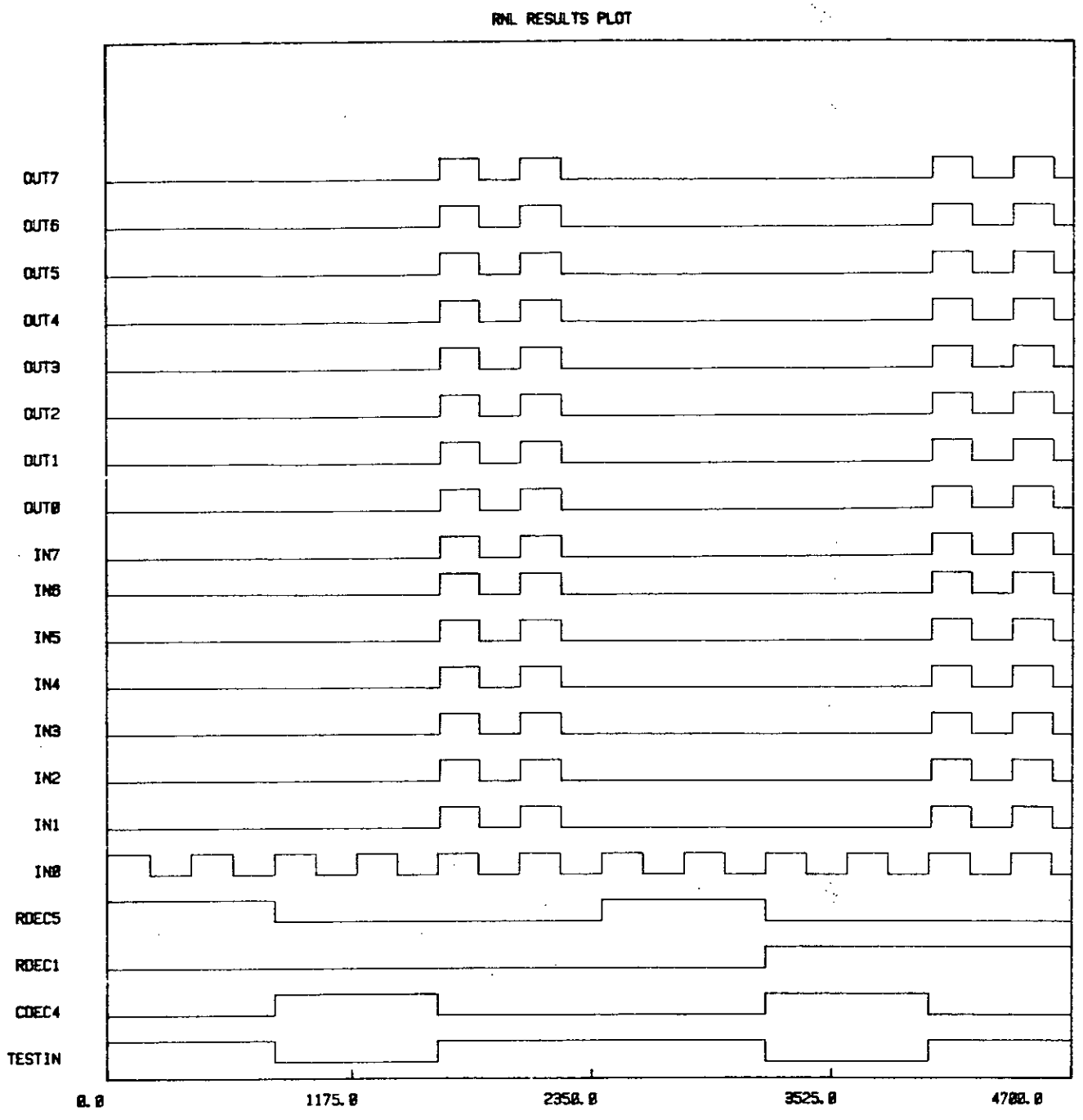

Figure 4.5 Switch level simulation of the test scheme for the MCE chip

matrix row where there is no other S_ON switch cell. The response also indicated ten switch cells with an S_ON fault, illustrated by the appearance of four pulses during the period of time when "TEST" (RDEC5) is set to logic zero. For three matrix rows (first, fifth, and last) with no signal present, another input point and, therefore another

testing cell, was tried for the test signals. The result for this test produces exactly the same fault conditions in these three rows as detected before. It is fairly certain that the row decoder for these three rows does not work properly. Figure 4.7 shows an output of a different MCE chip. There is no S_ON fault in the matrix, instead, there are quite a few S_OP faults. Similarly, there are three lines from the row decoder which do not function. An interesting finding is that the output from "IN4" has disappeared. This is caused by the faults causing an absent logic one for that column, or the faulty "Tesc" circuit for that column. This fault can be mapped on to the matrix as all the switch cells on that column are S_OP. From these results, a fault pattern for each chip can be generated. This pattern is particularly useful when message routing is going to be performed within the switch matrix. Such message routing will be discussed in the Chapter 5. Figures 4.8 and 4.9 show the two fault patterns for those two matrices, whose test results have been shown in Figures 4.6 and 4.7. Figure 4.10 shows a similar test applied to a Plessey chip with duplex matrix organisation. The Plessey duplex matrix chip is much bigger than the MCE one, therefore under the same test scheme, the "testing cell" has to drive a much larger capacitive load than that in the MCE chip. This results in a poor output response, causing some missing pulses in the DAS output shown in Figure 4.10. By using an oscilloscope, these missing pulses were observed with smaller magnitudes. Figure 4.11 illustrates an output of a Plessey chip with the normal matrix organisation. Eight response signals are shown here. Eight matrix rows were tested one-by-one, addressed by the signals RDEC1-RDEC6. Compared to the expected output response shown in Figure 4.3, Figure 4.11 indicates that those switch cells under test are functional.

By testing all the sample chips, the yield was found to be unexpectedly low for the MCE chip. Some recognisable processing defects in the MCE chip have been found under SEM examination. For example, Figure 4.12 illustrates three metal lines shorted together. Figure 4.13 shows a short between two metal tracks in a chip. Other potential defects, such as pinholes, which may be present, can not be easily observed under normal practical examination. For such defects, all the layout patterns may look correct, but logical faults can exist. However, the yield of the Plessey chips seem very good. Due to the limitation of the test facilities, such as the limited number of channels for pattern generation and data acquisition, several portions of the chips were randomly selected for testing. It was shown that most of the chips near the centre of the wafer are functional, with few defective switch cells.

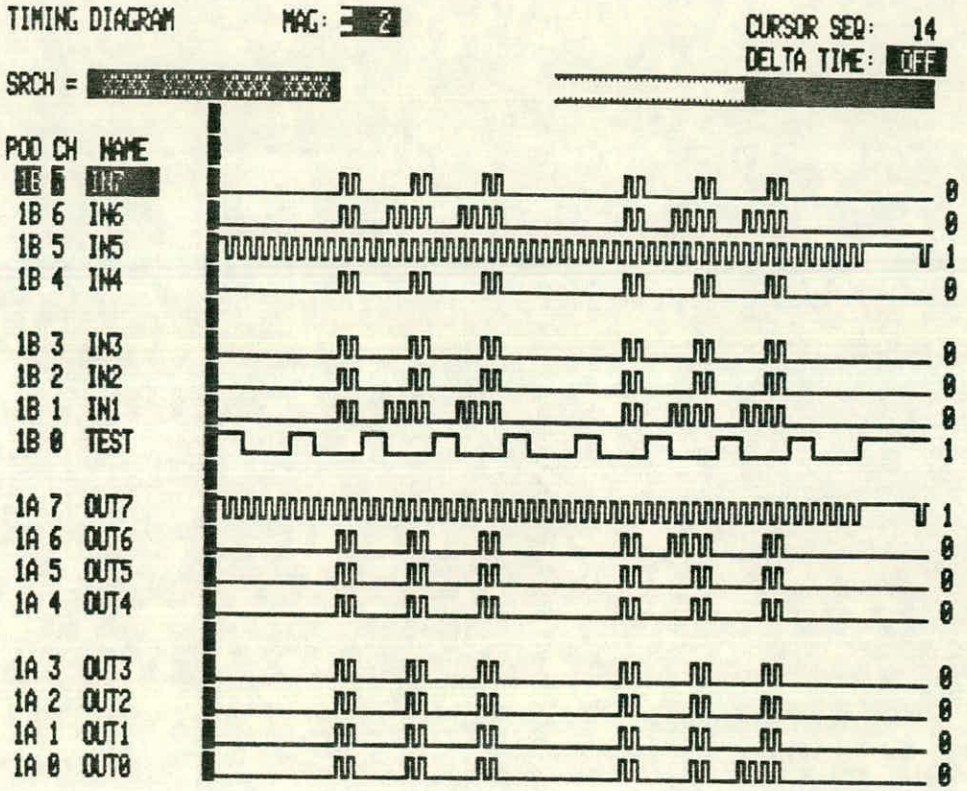

Figure 4.6 Test output of an MCE chip showing some S_ON switch cells

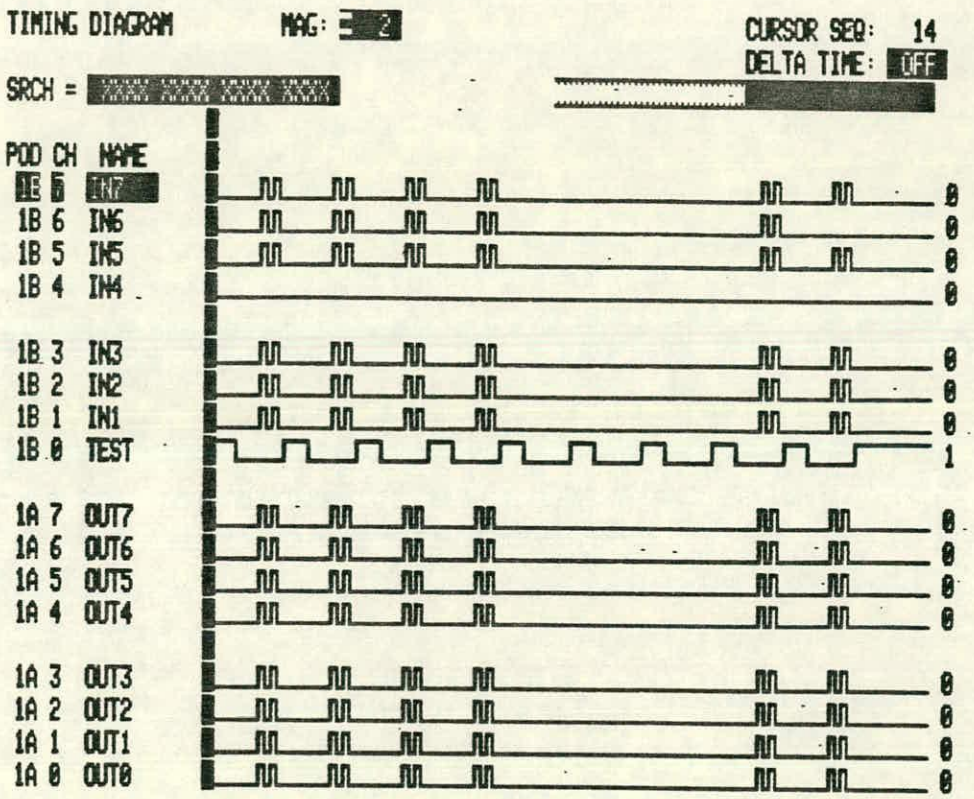

Figure 4.7 Test output of a different MCE chip showing some S_OP switch cells

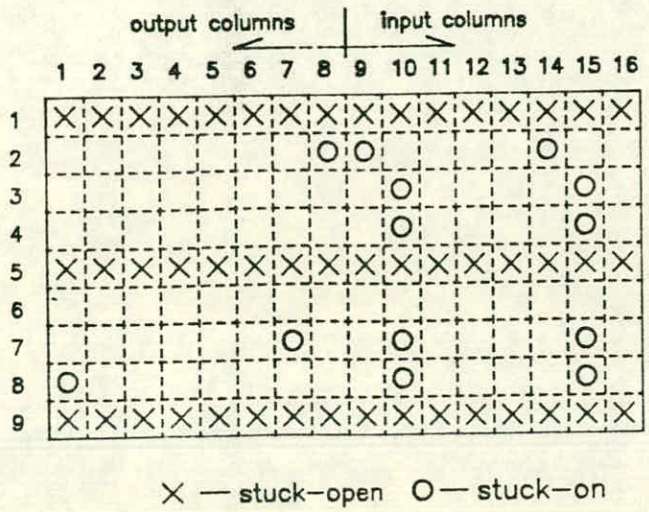

Figure 4.8 Fault pattern derived from the test result in Figure 4.6

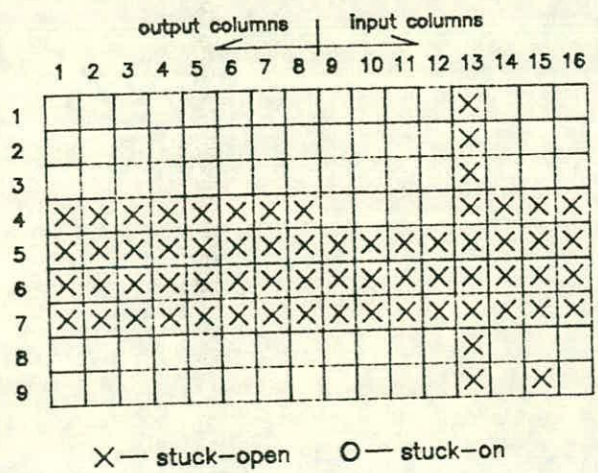

Figure 4.9 Fault pattern derived from the test result in Figure 4.7

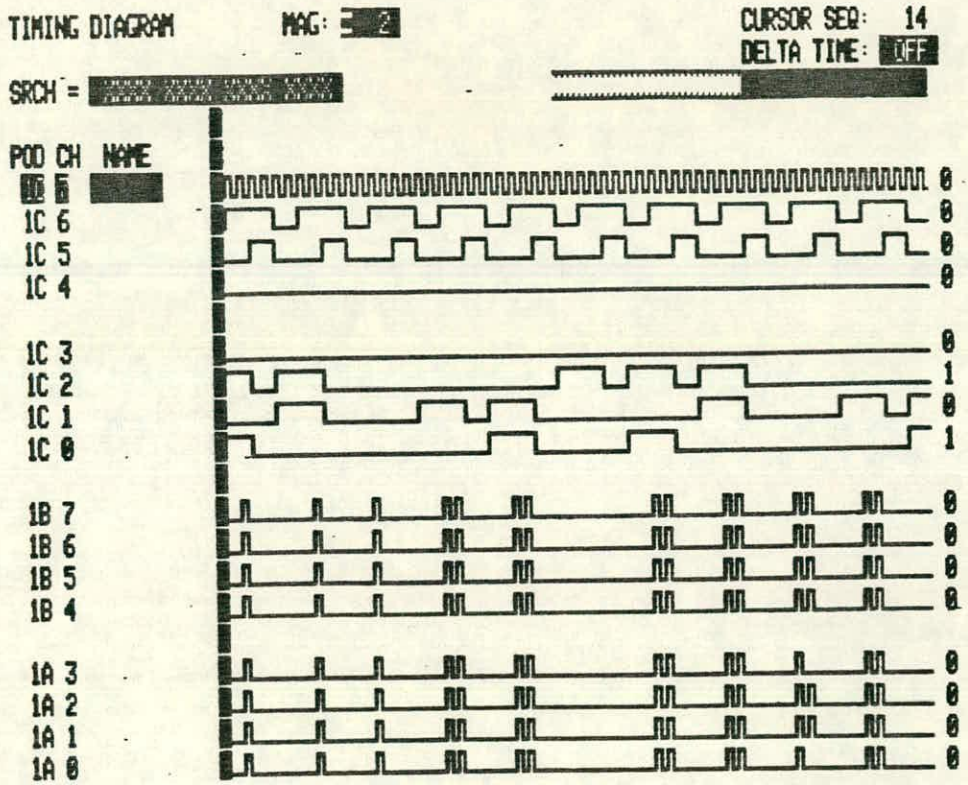

Figure 4.10 Test output of a Plessey chip with the duplex matrix organisation (six matrix rows tested).

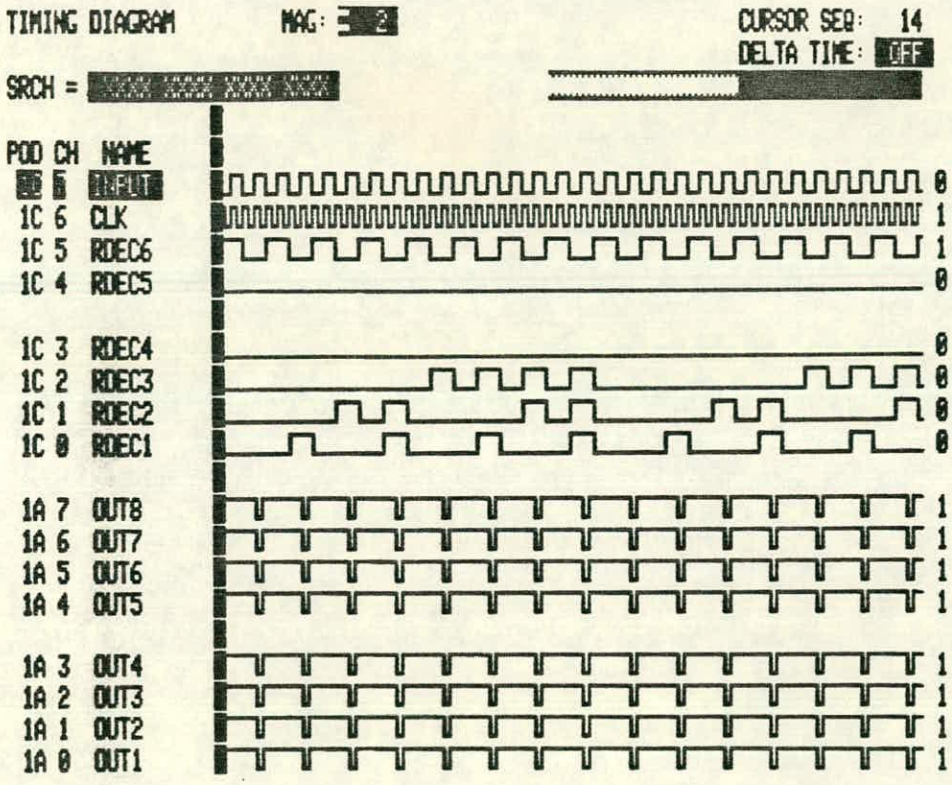

Figure 4.11 Test output of a Plessey chip with the normal matrix organisation (eight matrix rows tested).

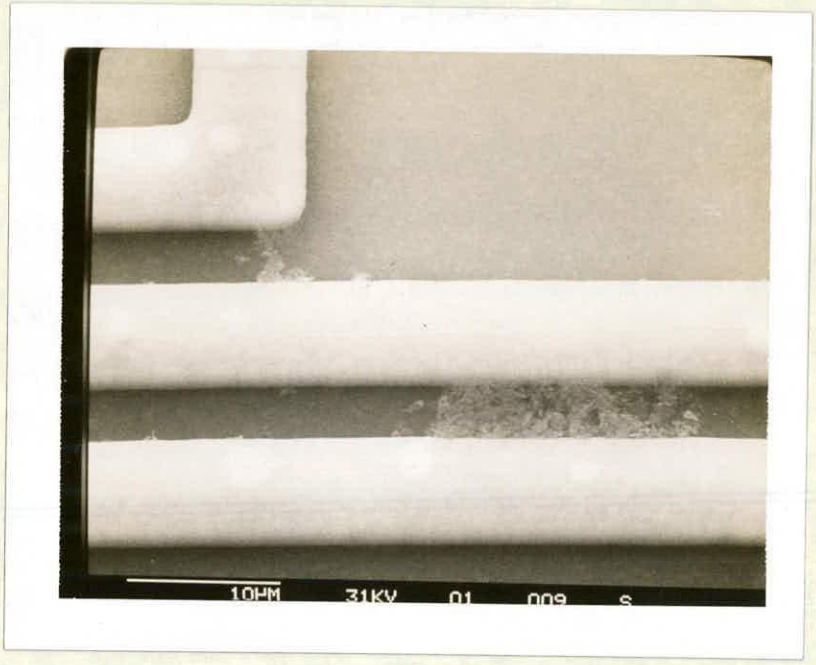

Figure 4.12 An SEM photo showing that three metal tracks are shorted to each other.

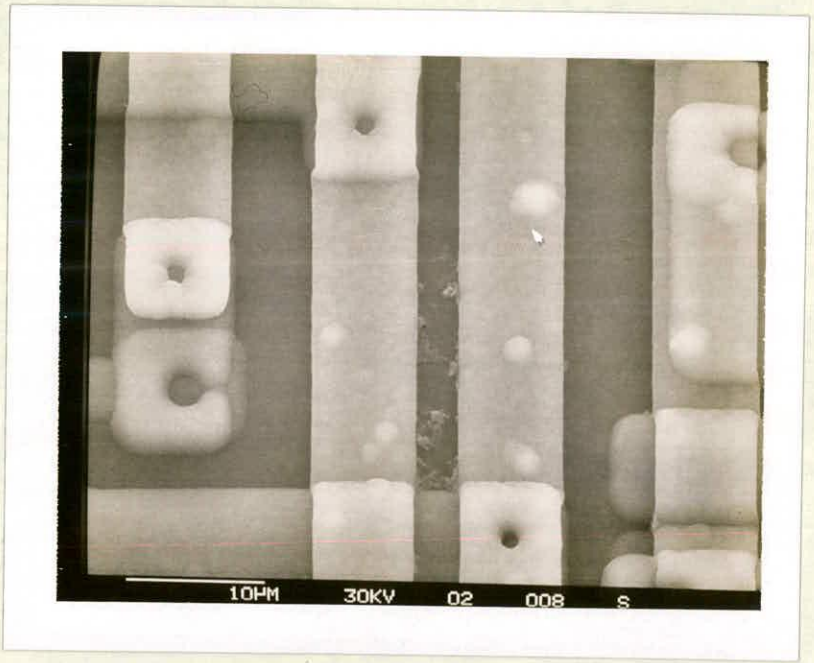

Figure 4.13 An SEM photo showing that two metal tracks are shorted to each other.

4.4.2. Characterising the prototype

To evaluate the characteristics of the switch matrix used in transmitting signals from the input to the output of the matrix, one route was selected for examination†. Furthermore, the correctness of the loading scheme developed earlier in Chapter 2 can be examined. Figure 4.14 shows how an output follows an input, subsequent to a connection being established within the MCE duplex matrix chip. Referring to the loading signals shown in Figure 4.2, the column address is represented by one bit in the diagram to indicate the timing of the column addressing, which is the same as for the row address signals. Before addressing the route, the entire matrix is initialised by setting both decoders to the "clearing state". Thereafter, the loading signals, shown Figure 4.2, are applied to the matrix. The signals are changed at different times to avoid possible race conditions, and left long enough for the matrix to settle down after initialsation. The output response validates the correctness of the loading scheme. It also proves that the prototype design is correct.

The same loading signals as shown in Figure 4.1 were applied to all the prototype chips. Figures 4.15 – 4.18 show four output responses from four different MCE chips. As compared to Figure 4.1, three chips have only one route correctly established under this particular addressing arrangement (OUT7, OUT6, and OUT5 respectively), and one chip has four routes correctly established (OUT0, OUT1, OUT4, and OUT7). Figure 4.19 illustrates an output response from a Plessey chip with the duplex matrix organisation, in which 8 signal paths were established one after another. Compared to the simulation result in Figure 4.1, the portion of the chip under test is fully functional. Figure 4.20 shows an output response from a Plessey chip with the normal matrix organisation, in which 8 signal paths were established one after another. The loading scheme shown in Figure 2.16 was used to establish a signal path. Again, the portion of the chip under test is fully functional.

† Which particular route is selected in the matrix is not important, because the composition of all the possible routes in the matrix are electrically equivalent.

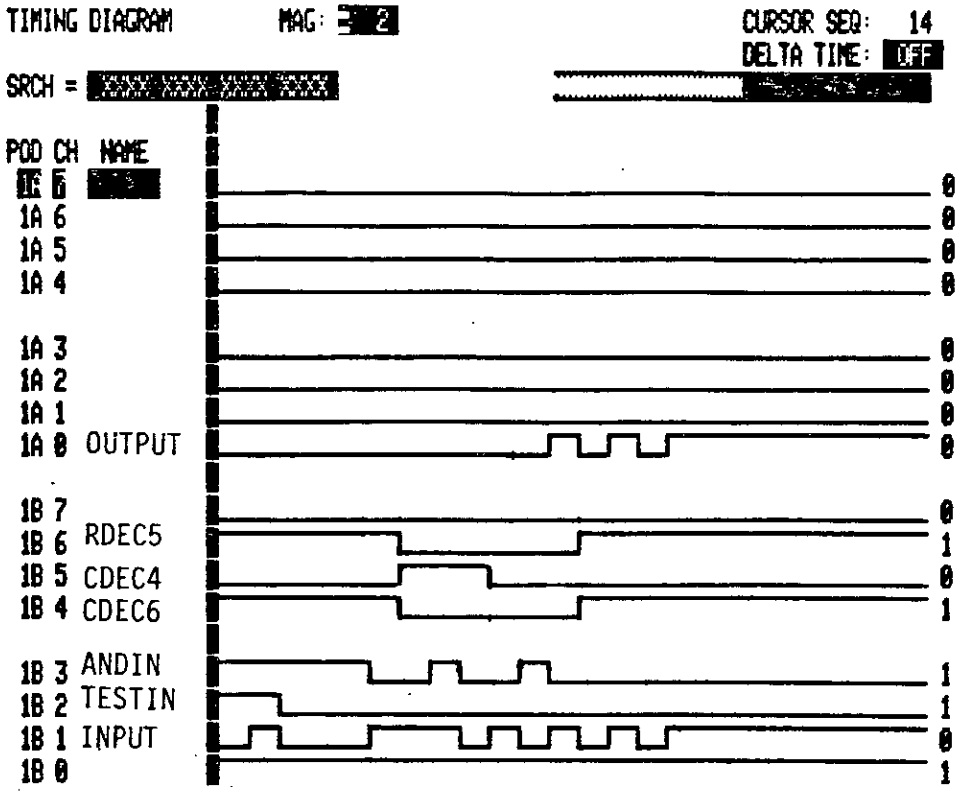

Figure 4.14 A routing output showing the process of establishing a physical path in the MCE chip.

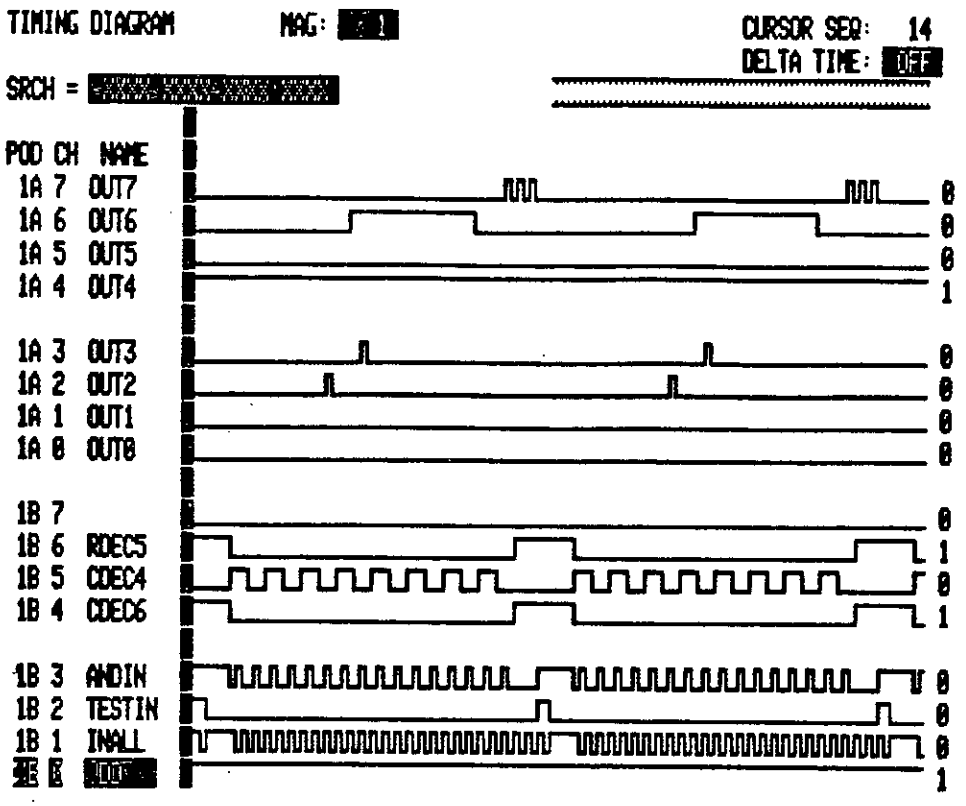

Figure 4.15 A routing output from an MCE chip showing that the output from OUT7 is correct.

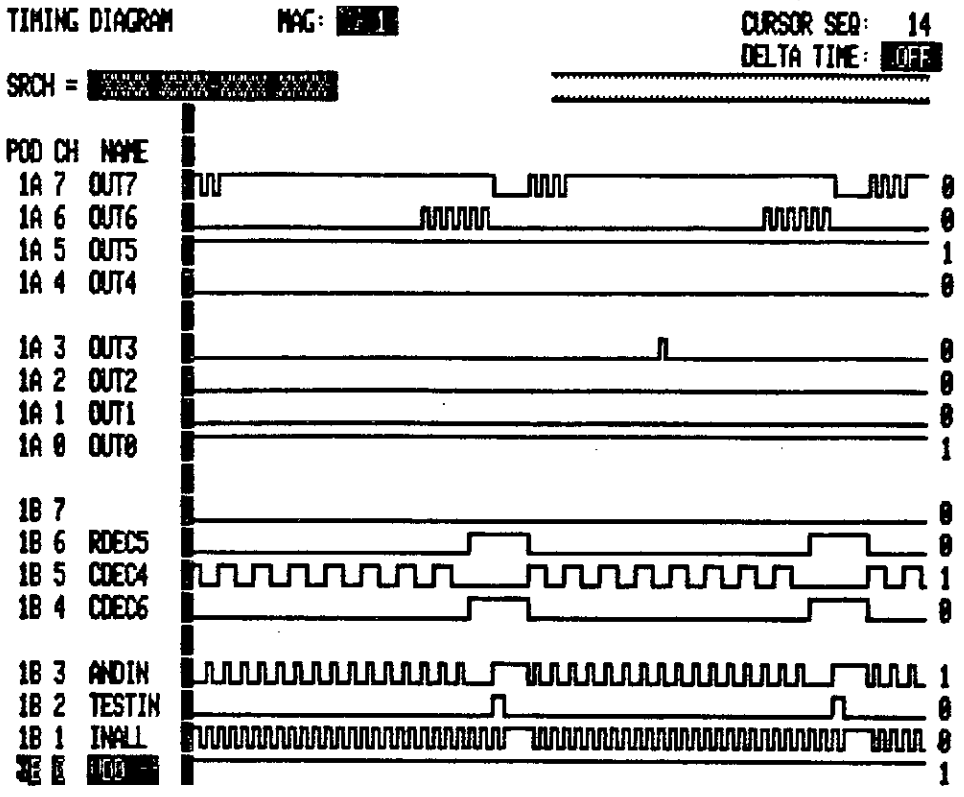

Figure 4.16 A routing output from an MCE chip showing that the output from OUT6 is correct.

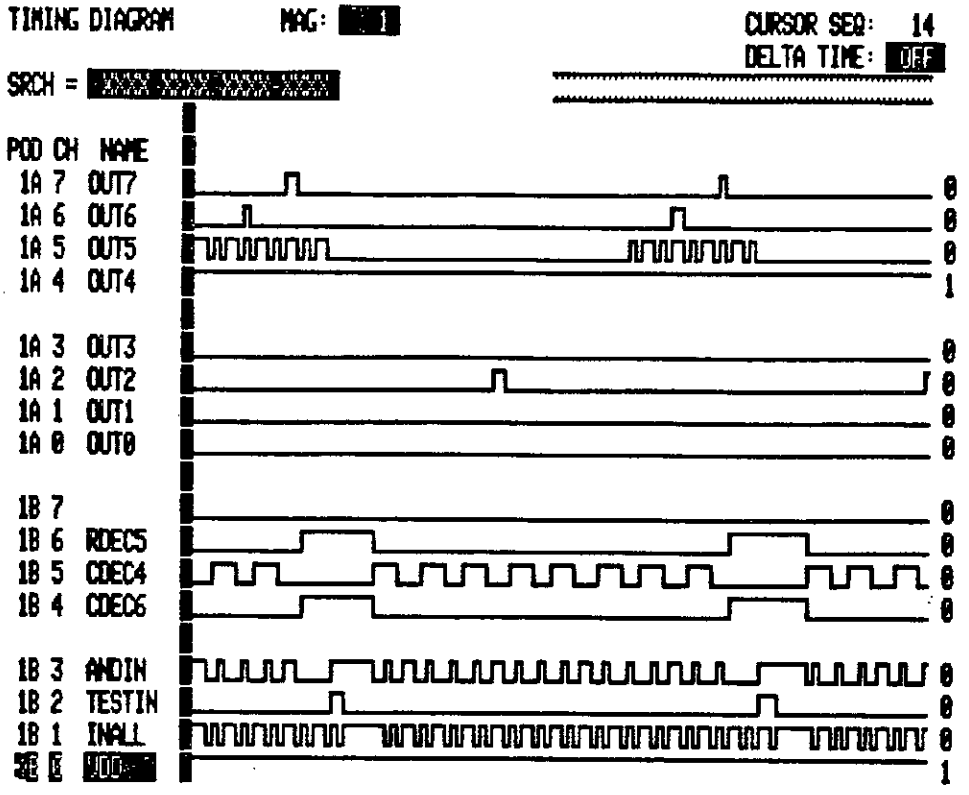

Figure 4.17 A routing output from an MCE chip showing that the output from OUT5 is correct.

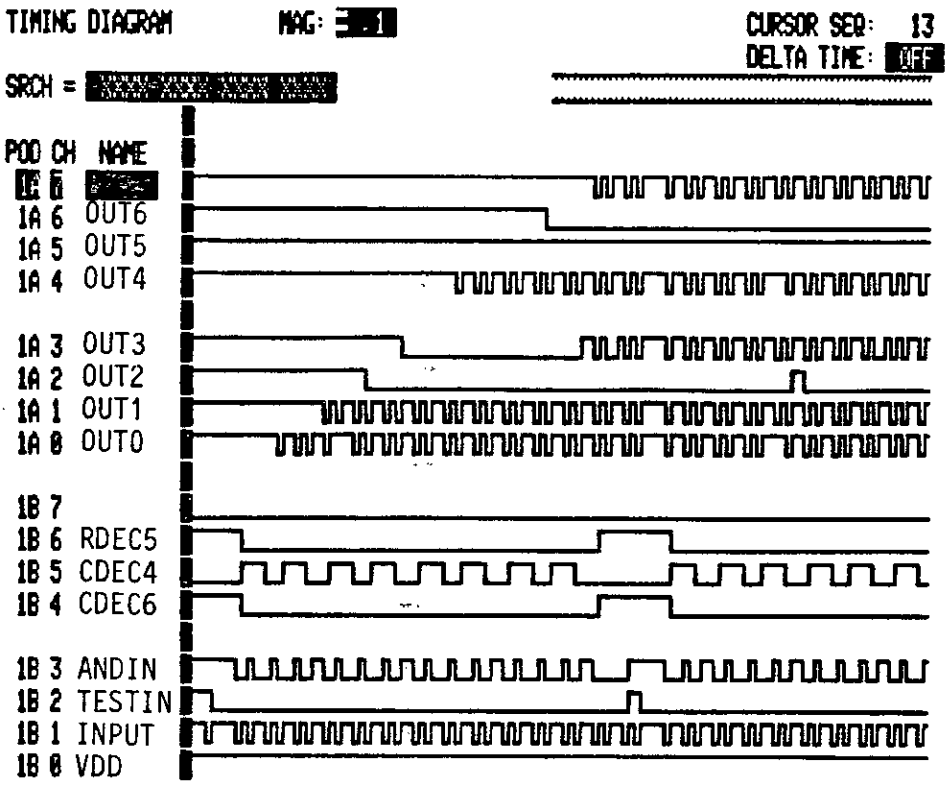

Figure 4.18 A routing output from an MCE chip showing that the outputsfrom OUT0, OUT1, OUT4, and OUT7 are correct.

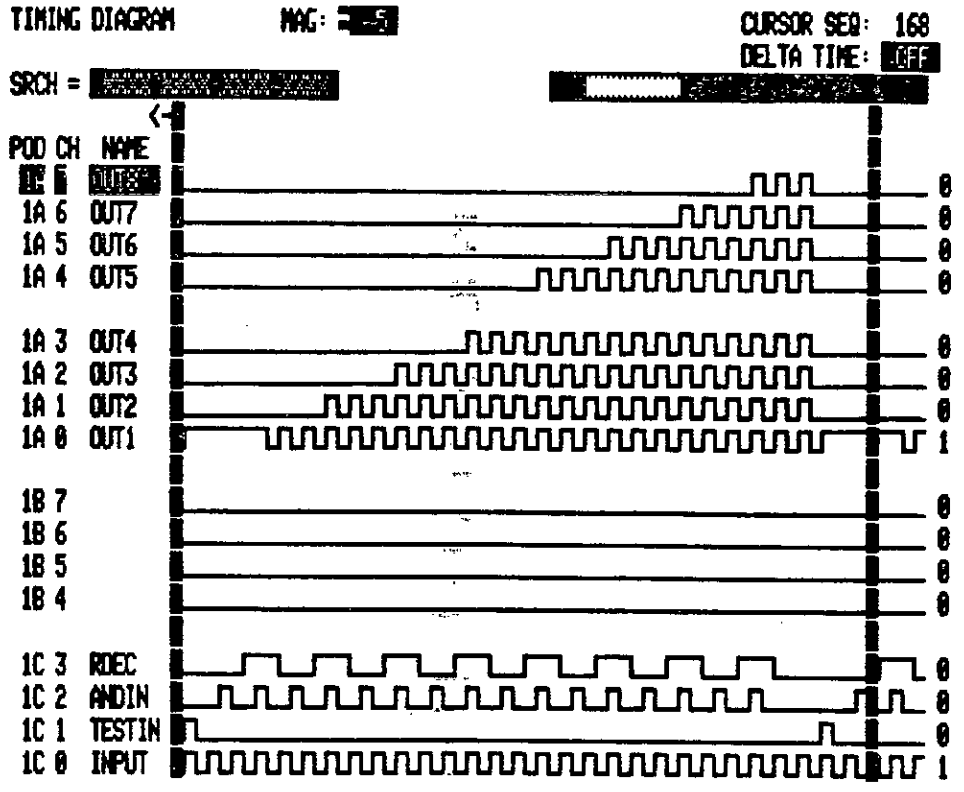

Figure 4.19 A routing output from a Plessey chip with the duplex matrix organisation showing that all the eight signal paths under test are functional.

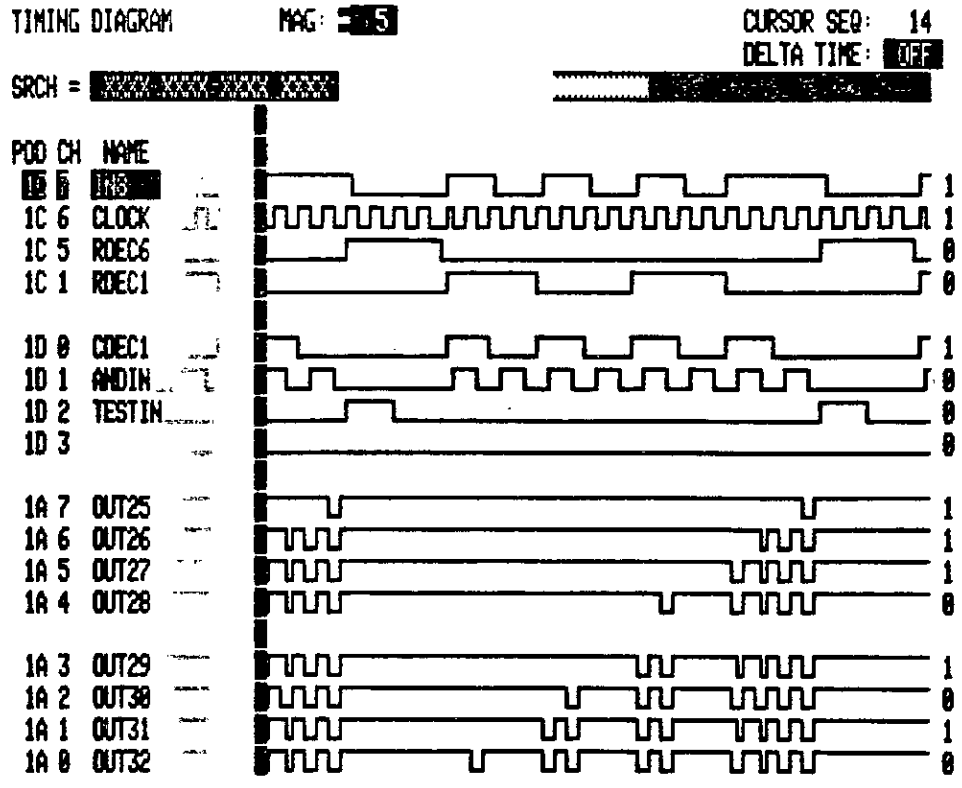

Figure 4.20 A routing output from a Plessey chip with the normal matrix organisation showing that all the eight signal paths under test are functional.

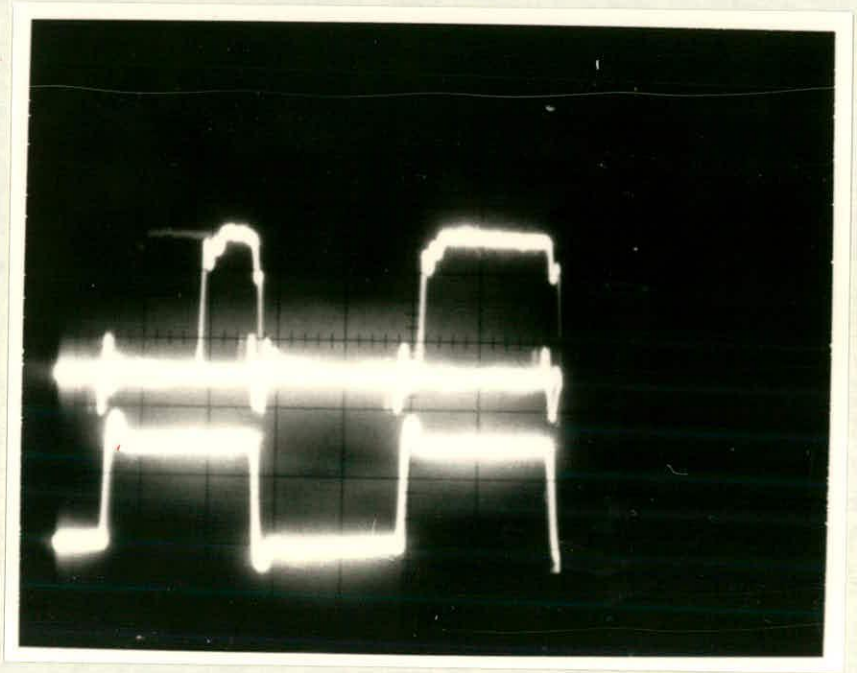

Figure 4.21 First two pulses of an output waveform from an MCE chip after a signal path is established (Time = 200ns/DIV, Voltage = 2V/DIV).

To characterise the quality of signals transmitted by the switch matrix, these output responses were examined using an oscilloscope. For the MCE chip, Figure 4.21 shows the waveform of the first two output pulses after the connection is established. The output signal is at the top and input signal is at the bottom. The waveform shows that there is a considerable amount of rising delay on the first output pulse. This occurs because, at the rising edge of the first input pulse, the physical route has not actually be established. In fact, the rising edge of the second enabling signal of "Dset" also starts appearing at the same time as that of the first input pulse. The switch cell will take time to be turned on, and complete the physical connection. Thus the input signal waits for the connection to be established, during which time the output does not change with the input. After about 240ns, the output starts responding to the changes in input signal, following the physical connection being established. This event has already been depicted in the switch level simulation result shown in Figure 4.1, in which all the first pulses of the output signals are relatively smaller than the rest of the output pulses. The actual risetime of a signal caused by the switch matrix when it goes through can be obtained from the second output pulse, where the physical path has already been established. Figures 4.22 and 4.23 show both the rising and falling edges of both the input

and the output respectively (on the expanded scales to those in Figure 4.21). The risetime is shown to be about 60ns and falltime is about 30ns. To validate these results, a complete signal path was extracted and simulated from the prototype chip using MEXTRA and SPICE. The information extracted includes all the transistors involved in the path, all the stray capacitances attached to the related nodes and related transistors, which are not involved in this logic function generation but do contribute junction capacitance to the path. The stray capacitances are calculated for the whole chip. Figure 4.24 illustrates the equivalent circuit extracted in the simulation. The resistor appearing in the circuit is the protection resistor incorporated at the input pad. The first buffer (inverter) is an intermediate stage buffer in the chip; the second buffer is the output buffer at the output pad. It was observed that the output voltage level of the test patterns from the DAS is about 3.5V for a logic one, even when the voltage supply is 5V. This is caused by the voltage drop on the test probes. The equivalent circuit incorporating this effect has been simulated, and Figure 4.25 shows the output. It indicates that the risetime is about 50ns and the falltime around 30ns. These figures are close to the data obtained from the test, with the difference resulting from either the inaccuracy of the parameters employed in the circuit extraction or an unexpected parasitic capacitance and inductance in the test environment. The longer risetime is mainly caused by the lower voltage level of the input signal. In practice, when the voltage level is low, the p-type transistors in the transmission gate paths will not be in a strong conduction state. Consequently, an increase in the voltage level of the input signal was tried during the test by increasing the supply voltage above 5V. As a result, the risetime reduced and falltime increased. The reason for the falltime increasing is caused by the internal nodes storing more charge now than in the previous case, so that the discharge time will be longer. Figure 4.26 shows a simulation result of the extracted path circuit with the ideal 5V input signal swing. This shows that the input signal will potentially have about 30-40ns risetime and falltime through the switch matrix, with very little signal distortion (i.e. variation of the pulse width). Figure 4.27 shows the input and output waveform of a Plessey chip with the normal matrix organisation. The output falltime is 100ns, and the output risetime is 50ns. Figure 4.28 shows the input and output waveform of a Plessey chip with the duplex matrix organisation. The risetime and falltime of the output are about 50ns.

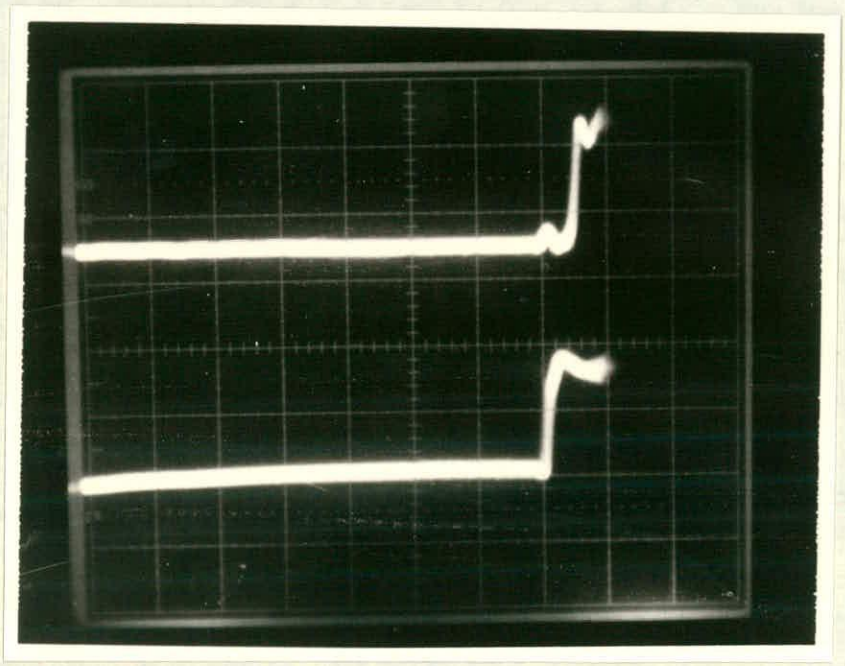

Figure 4.22 Expanded waveform from Figure 4.21 showing the rising edge (Time = 100ns/DIV, Voltage = 2V/DIV).

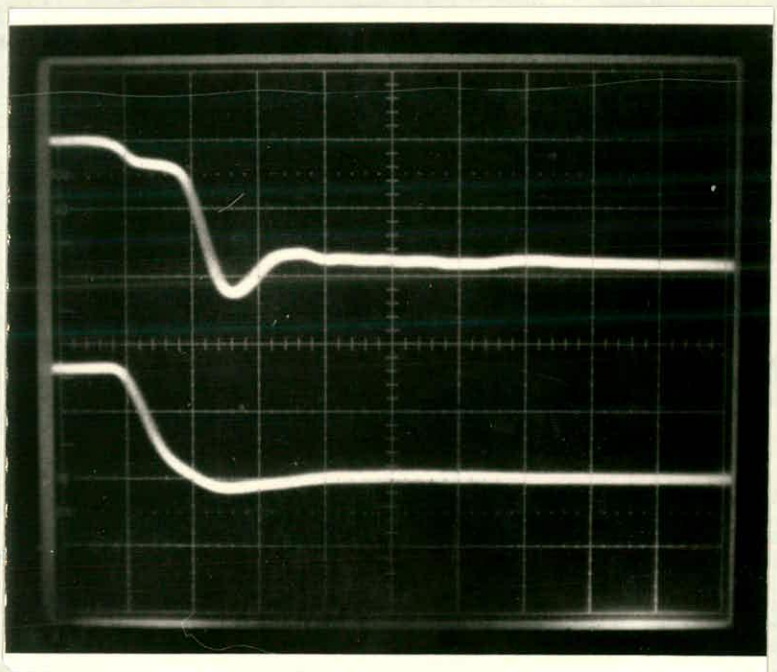

Figure 4.23 Expanded waveform from Figure 4.21 showing the falling edge (Time = 20ns/DIV, Voltage = 2V/DIV).

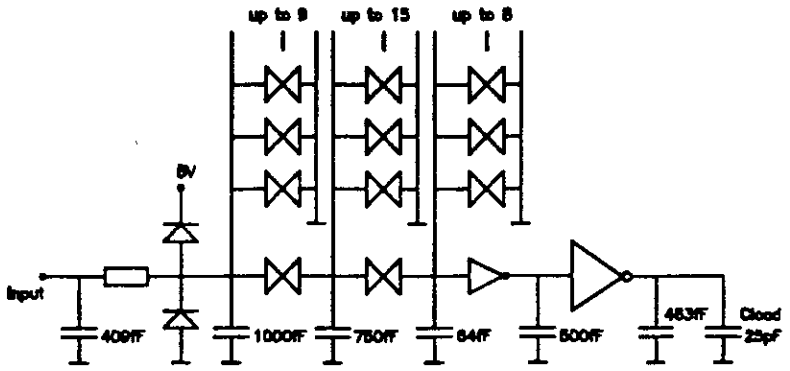

Figure 4.24 Equivalent circuit of a single path in the MCE chip.

4.5. Design Deficiencies

A problem encountered during the test was that the fault pattern generated from the response signals to the test pattern does not include information on possible faults in the column decoder. This happens because all the control for switch-setting during the testing are set by "Tset" rather than the column decoder itself. This problem was particularly serious during the MCE chip testing, because the chips are more defective than the Plessey chips. For example, the fault pattern in Figure 4.8 shows that five possible connections can be made in the matrix. However, the result is disappointing. Some faults exist in that chip in both the column decoder and the NAND gate. Figure 4.29 is an SEM trace which illustrated a fatal fault in the loading control signals for the column addressing in the matrix. The bright line in the picture indicates a logic one. Only the column decoder is stimulated here by applying addressing signals to its inputs CDEC1–CDEC6. "Dset" is set to logic one and "Tset" is set to logic zero, so that the loading control signals for matrix columns can go through the NAND and Tesc circuit before reaching the switch cell array. The picture also shows that outputs from the column decoder are stopped at the NAND gate and can not reach the array. Therefore, the switch cells on these columns can not be properly turned on. Figure 4.30 shows that the same stimulation applied to another chip with working column decoder, NAND gates, and "Tesc" circuit. The long bright horizontal lines on the right hand side show that the loading control signals from the column decoder reach the switch cell array.

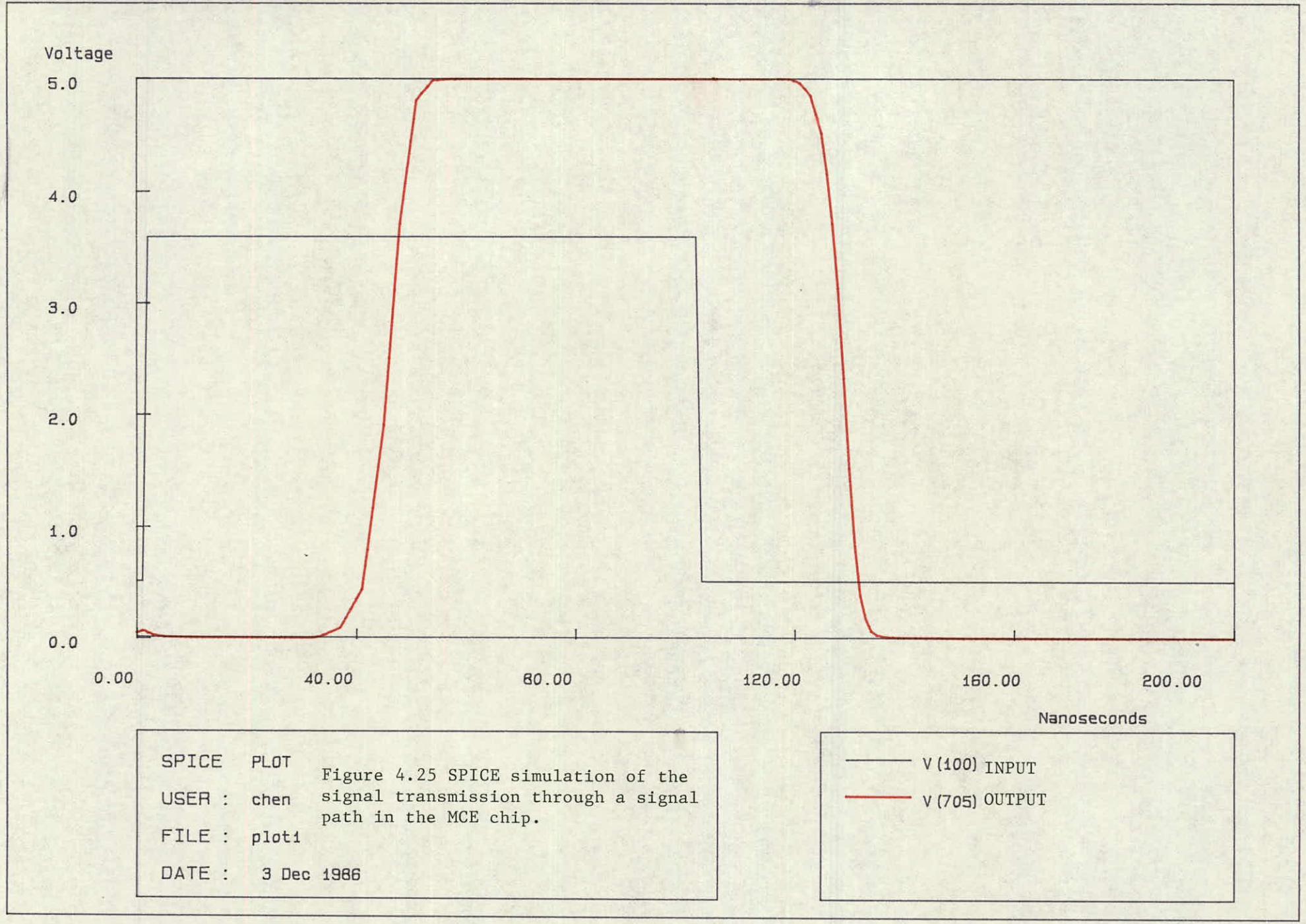

Figure 4.25 SPICE simulation of the signal transmission through a signal path in the MCE chip.

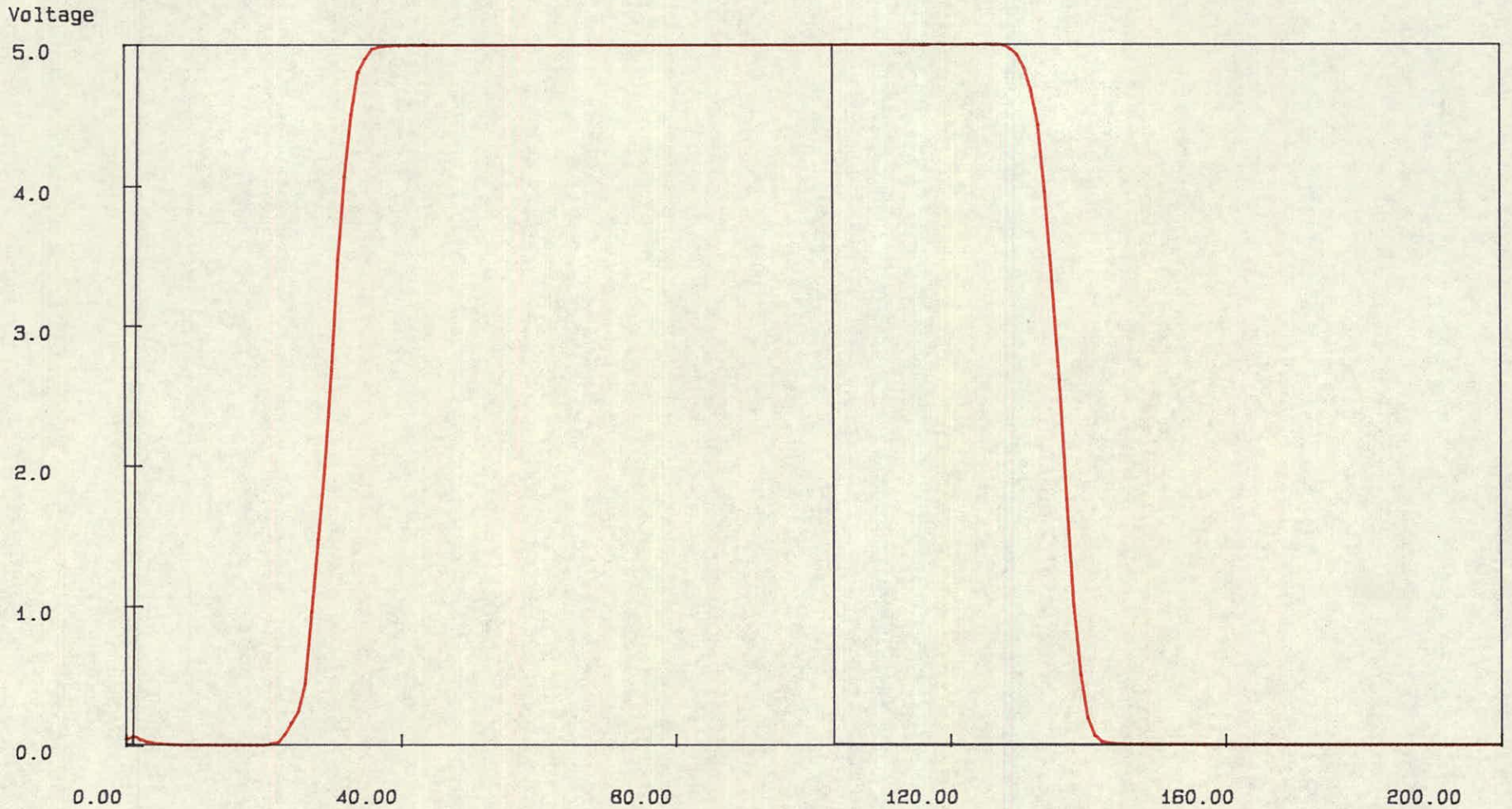

Figure 4.26 SPICE simulation of the signal transmission through a signal path in the MCE chip with full input voltage swing

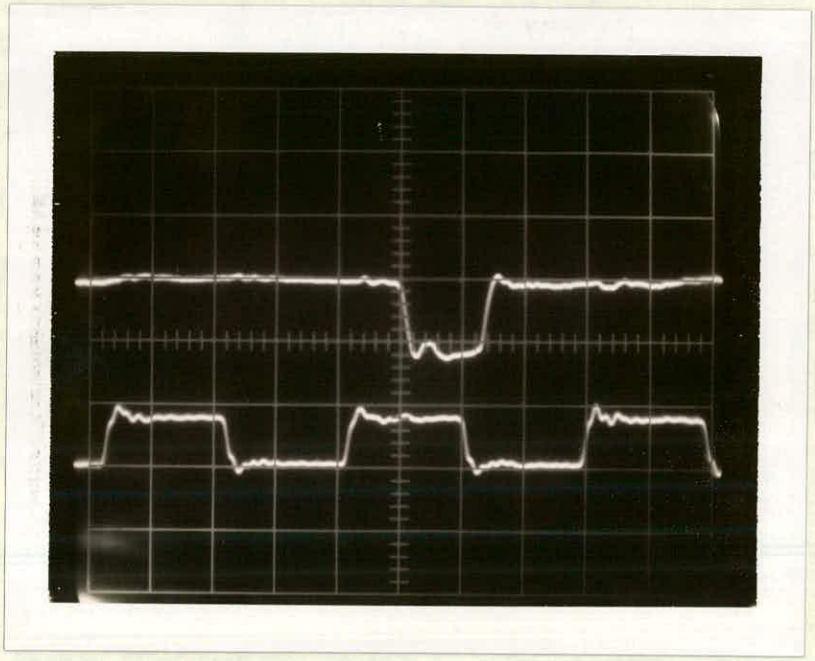

Figure 4.27, Input (bottom) and output (top) waveform of a signal through a path in a Plessey chip with the normal matrix organisation (time = 100nsec/DIV, voltage = 5V/DIV).

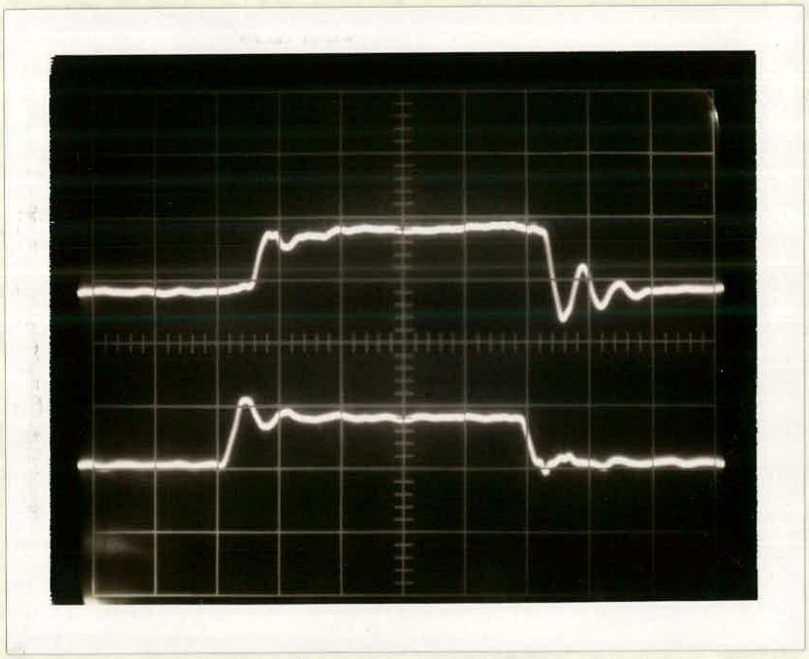

Figure 4.28, Input (bottom) and output (top) waveform of a signal through a path in a Plessey chip with the duplex matrix organisation (time = 100nsec/DIV, voltage = 5V/DIV).

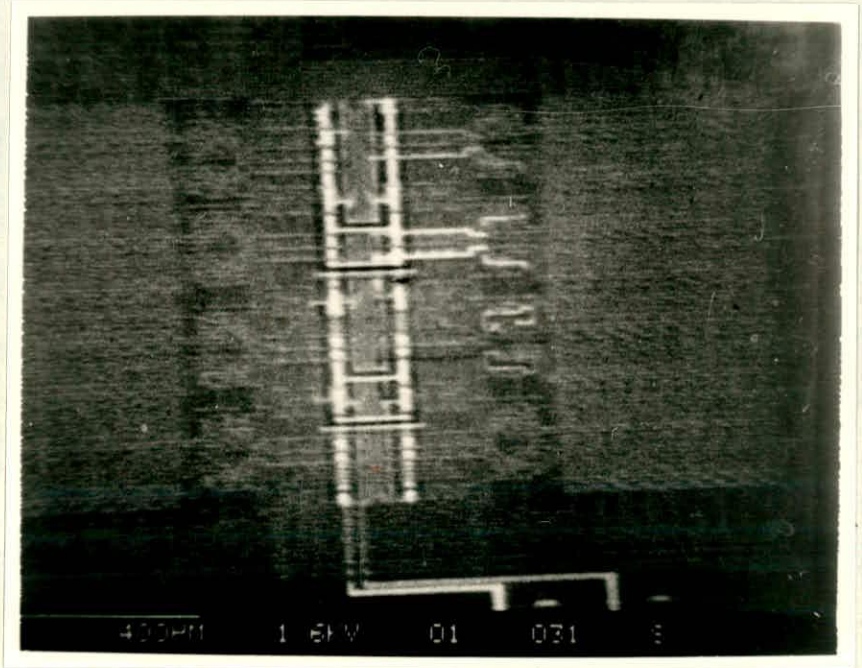

Figure 4.29 An SEM photo showing a fault in the NAND gate circuits in an MCE chip.

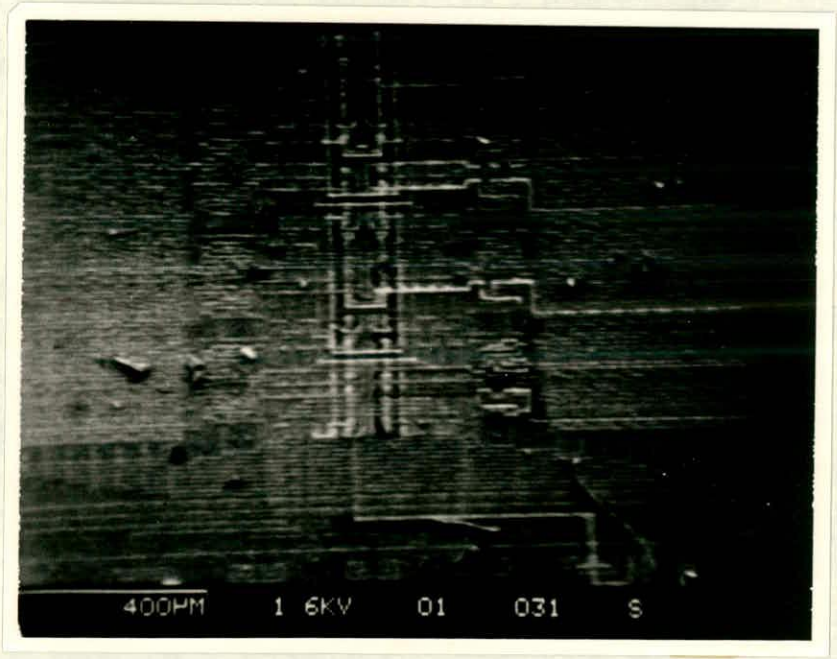

Figure 4.30 An SEM photo showing a working that the column decoding circuits in an MCE chip are functional.

In the present chip organisation it is difficult to test the column decoder efficiently. Additional circuits need to be introduced into the chip to implement a satisfactory test for the column decoder. For example, a chain of shift register stages could be placed between the column decoder and the switch cell array. In the test mode, these shift registers fetch outputs from the column decoder and shift them out. In the working mode, they simply bypass the signals. Such a scheme does not add a considerable amount of extra I/O pins to the chip. Details of such a modification will be presented in Chapter 7.

4.6. Summary

This Chapter presents a switch matrix prototype design and its associated test results. The correctness and characteristics of the prototype have been thoroughly investigated, although the yield is unexpectedly low. From the results obtained, it is strongly felt that the control circuits, which do not have a defect tolerant capability, should be designed with more conservative design rules. This should result in less sacrifice of some rows and columns of working switch cells. The design has also been criticised for having difficulties in detecting and locating faults in the column decoder. This should lead to necessary modifications of the prototype, such as will be summarized in Chapter 7. The test results from this Chapter, particularly the fault pattern, lay down the basis for the discussion in Chapter 5 of automatic message routing within the switch matrix.

CHAPTER 5

A MESSAGE ROUTING ALGORITHM FOR THE CROSSBAR SWITCH MATRIX IN THE PRESENCE OF FAULTY SWITCHES

5.1. Introduction

In Chapter 3 fault models for the switch cells were established. In Chapter 4, a method of generating a fault-pattern for a particular switch matrix by testing the switch cell array, decoders, etc. was produced. Even after all the processing elements (PEs) in the superchip have been tested the problem remains of how to use this superchip, and search for a connection set within the crossbar switch matrix to correctly connect the working PEs together to form a desired system. This is not a trivial task, particularly when the system is large, as there may be many interconnections, and the switch matrix is not perfect. An algorithmic method for searching the interconnection routes within a faulty crossbar switch matrix is presented here. In this Chapter, the term *fault pattern* has a slightly different meaning from that used in Chapter 4. Unless otherwise stated, it here refers to the overall fault pattern including the fault pattern of the switch matrix, as well as the fault pattern of the PE bank.

5.2. The Problem and Solution Tactics

Unlike routing problems in some other types of interconnection network [127, 128] where packet switching conventions are assumed, the routing problem in the crossbar switch matrix relates to circuit switching, so as to establish a physical path between one source and one or more destinations. As in the system customisation procedure for bit-serial systems proposed earlier, where all the PEs are linked together to form a system via the switch matrix, in this work the routing implementation must be sure of 100% realisation for all the desired interconnections within the matrix. Any failure in establishing a physical connection will cause the total failure of system operation. This is

conceptually different from a conventional multiprocessing system which does not necessarily need 100% realisation of communication at any time. If the switch matrix is perfect, then the routing is very straightforward and 100% routability can always be achieved. But, unfortunately, as seen in Chapter 4, processing faults are inevitable. Therefore, the routing problem becomes one of how to tolerate the existing faults in the matrix, but still implement all the desired interconnections. In other words, an efficient way must be found to avoid or utilise defective switch cells in the matrix in a maximal way. As discussed in Chapter 3, the switch cells in the matrix may be faulty in one of four ways: S_OP, S_ON, SA1, and SA0. SA1 and SA0 faults cannot be tolerated within the matrix. The whole matrix row or column will be sacrificed where SA1 or SA0 appears. S_OP and S_ON faults may be tolerated within the matrix either by avoiding them or utilising them. For instance, switch cells with S_ON faults at the right place can be used to establish interconnections.

To implement a set of interconnections within a switch matrix correctly under the presence of some switch failures, several factors must be known:

- the specification of the system organisation which determines all the desired interconnections to be routed through the switch matrix.
- the fault pattern of the superchip, indicating locations of all the failed PEs and switch cells as well as the failure modes: S_OP or S_ON, of the latter.
- chip composition information which gives the locations of all processing elements (either working or faulty) in the chip.

A net-list of the system can be derived from the specification of the system organisation. Because of the switch matrix structure, each matrix row will and can only accommodate one net. A system with m nets should at least have m matrix rows. Deciding which matrix row holds which net is a non-trivial task. The outcome of this study will directly determine the success of the whole routing procedure. Conceptually, this is similar to a so-called *Linear Assignment* problem. To better understand the matrix row−net assignment problem, the linear assignment problem is illustrated first. The following example shows how such a problem is composed.

Example:

Suppose we have five men and five jobs to do. Each man has a skill level at each possible job. We are also constrained not to put more than one man on each job. According to the different skills, assigning different men to a job will have different cost. Assume that we have a cost matrix shown in Table 5.1, in which "4" means this man can not do that job at all. The problem is to find a best (the most economic) way to assign each man to these five different jobs. We may find one solution of this problem by asking the 1st man do the 1st job, the 2nd man do the 4th job, the 3rd man do the 3rd job, the 4th man do the 5th job and the 5th man do the 2nd job.

Table 5.1 Costs of the Job Assignment

	1st Man	2nd Man	3rd Man	4th Man	5th Man
1st job	1	4	2	3	3
2nd job	2	2	2	2	1
3rd job	4	3	2	2	3
4th job	3	2	3	3	2
5th job	1	1	2	1	1

There are various algorithms for optimal linear assignment. Munkres [129], Hannan and Kurtzberg [130] have defined and discussed the problem. Many of the algorithms are variants of Kuhn's method (also known as the Hungarian method). In the above example, assessing the cost value of each different man for each different job is crucial to the whole process. This measure relates algorithmic domain (how are the jobs allocated?) to the situation in the physical domain (is it economic?). The solution will directly affect the cost-effectiveness of the assignment. Back to our routing problem, instead of having a cost value for the matrix row−net assignment, a goodness measure will be taken as a measure for row−net assignment. This is complementary to the cost. The relationship of each assignment and its goodness value is given by a *goodness function*. The goodness function for the routing problem will be discussed in Section 5.3. Here, let us suppose that the goodness value for each possible matrix row−net

assignment has been given. These goodness values are not put in a matrix form as in the above example. Instead, all the goodness values related to one net, i.e. all the possible goodness values of this net being assigned to every matrix row in the switch matrix, are put into a queue which will be sorted to form a priority queue for this net. This priority queue indicates the best choice of assigning this net to some matrix rows. The overall goodness value for the system is the sum of all the individual goodness value associated with an assignment set. According to the priority queues and the actual goodness values in the queues, assign each different net to a different matrix row with the highest possible overall goodness value. Then try to make the system interconnections with the assignment set. The goodness value, as shown later, is quantized. Situations can arise where the overall goodness value for the system is the same for a number of different assignment sets.

Definition 5.1:

> If two nets have the same goodness value (great than zero) for a matrix row, then a *conflict* occurs between the two nets. These two nets are called *conflicting nets*.

If a conflict happens during an assignment process, then one of the conflicting nets will be arbitrarily assigned to the matrix row, in the first instance, and at the same time the conflict information is recorded. If the routing with this assignment set does not succeed, then the assignments of the two conflicting nets are exchanged, and the whole routing is performed again.

Once a matrix row−net assignment is finished, the PEs specified by the user are implemented (or precisely allocated on to the PEs on the physical chip), one-by-one, together with the interconnection. If all the PEs are implemented and successfully connected to each other via the crossbar switching network, then the whole routing is regarded as successful. Otherwise, the assignment set is changed again by swapping a pair of conflicting nets, and then an attempt is made to implement the PEs again, until successful. If all the conflicting nets have been tried and the implementation has not been successful, then the system customisation on this particular superchip has failed.

5.3. Goodness Function for Net Assignment

An appropriate goodness function for any assignment is crucial to the success of finding system interconnections within the crossbar switch matrix. The goodness function proposed next is empirical, in particular with respect to the parameters chosen. It is the result of many experiments performed during this research, with different fault densities and different fault patterns. The goodness value can be positive or negative according to the number of faults and their locations in a particular matrix row. The more positive the goodness value is for a matrix row – net assignment, the better will be this net assignment for that particular matrix row.

Before commencing on the evaluation of the goodness function, some special notation is defined. Once every net has been assigned to a different matrix row, all the input and output ports of the PEs on a net must be connected to each other via certain switch cells on the associated matrix row. These switch cells link the ports of the PEs through the matrix columns, and are called *port-attached cells*, or "pat" for short. The switch cells on that matrix row, but not on these columns, are called *port-detached cells*, or "patb" for short. Combining this with two fault conditions, S_ON and S_OP, produces four cases: "S_ON_pat", "S_OP_pat", "S_ON_patb" and "S_OP_patb". The number of PEs of one kind (all the PEs required may not be identical) specified by the user as the system requirement is named as "uPCnt", and the actual number of the PEs of this kind on the superchip is named as "cPCnt". These could be different due to the introduction of redundancy of PEs in the superchip. Table 5.2 presents a summary of the notation adopted.

By examining the positive (good) side and the negative (bad) side of assigning a net to a matrix row, the following observations have been found by experiment:

Factors in favour of an assignment:

1. The number of S_ON cells on port-attached points (S_ON_pat) indicates the number of S_ON cells that could possibly be used to implement a net by the assignment. The larger the number of S_ON_pat, the better the assignment will be and, therefore, the higher the goodness value.

Table 5.2 Goodness Function Notation	
notation	**meaning**
S_ON_pat	stuck_on cell at port-attached point
S_OP_pat	stuck_open cell at port-attached point
S_ON_patb	stuck_on cell at port-detached point
S_OP_patb	stuck_open cell at port-detached point
uPCnt	number of processing elements of one kind required by the system
cPCnt	number of processing elements of one kind actually on the chip
uR_S_ON_patb	number of redundancies of the processing elements linked to the S_ON_patb cell currently being considered
uR_S_OP_pat	number of redundancies of the processing elements linked to the S_OP_pat cell currently being considered

2. The amount of redundancy available from PEs, to be abandoned† by some S_ON cells on port-detached points (S_ON_patb), illustrates how the damage caused by S_ON_patb cells in the assignment can be reduced by the availability of redundancy. The more redundancy, the less damage there will be to the customisation process, and consequently, the higher the goodness value for the assignment. However, relative redundancy should be considered here because it involves the complexity information of a PE. For example, if two different types of PE both have one redundant PE of its own type, but one serves 10 PEs of the same type, and the other serves only 3 of its type, then it is clear that the second type of PE is less defect tolerant and more complex (larger) than the first one. If a PE from both types has an S_ON_patb cell attached to a matrix row currently under consideration for the assignment, then the outcome should result in the abandonment

† An abandoned PE, no matter if it is defective or not, is inhibited from being used for the current routing trial.

of the PE of the first type (the smaller one). After much experimentation, the relative redundancy, γ, used in the goodness function is defined to be:

$$\gamma = \frac{A(uR\_S\_ON\_patb)}{(uPCnt + B^{(uR\_S\_ON\_patb)})} \qquad (5\text{-}1)$$

where A and B are two coefficients. A is actually a 'lifting' coefficient, whose meaning and value will be discussed later. Strictly speaking, relative redundancy should be the total redundancy divided by the total number of PEs of that type, i.e. $\frac{(uR\_S\_ON\_patb)}{uPCnt}$. However, by adding $B^{(uR\_S\_ON\_patb)}$ into the divider, the goodness value of the PE with fewer redundancies is increased as the above example illustrates. In other words, the goodness value for a larger PE is increased to prevent it being abandoned.

3. The amount of available redundancy for those PEs abandoned due to S_OP cells on port-attached points (S_OP_pat), indicates how the damage caused by the S_OP_pat cell in the assignment can be reduced by the availability of redundancy. The more redundancy, the less damage there will be to the customisation process and, therefore, the higher the goodness value for the assignment. The composition of this term is exactly the same as S_ON_patb, shown in equation (5-1).

4. The number of S_OP cells on port-detached points shows that these defective switch cells have been avoided in the assignment, and have no effect on the routing process. If an assignment creates many S_OP_patb cells, then it will probably make these cells not affect other assignments. Therefore, the more S_OP_patb cells present, the higher the goodness value will be for the assignment. Compared with other terms in the function, this is the least important factor in the reckoning. This follows because according to the experiments, S_OP cells do much less damage to the success of the routing process than S_ON cells. In other words, an S_ON_patb cell must be considered much more seriously. Its damage is far greater than the benefits brought by an S_OP_patb cell. Therefore, a small weight C is assigned to this term, C×(S_OP_patb).

Factors against an assignment,

1. The number of S_OP cells on port-attached points (S_OP_pat) indicates how much damage is due to an assignment. If there is any S_OP_pat cell in an assignment, then the PE linked to the column where this S_OP_pat cell is located has to be abandoned, because connection can not be achieved.

2. The number of S_ON cells on the port-detached points (S_ON_patb) shows how much damage there will be for the assignment. This has the same influence on the goodness value as that for S_OP_pat cells.

Combining these factors, a goodness function, G, follows:

$$
\begin{aligned}
G &= \Sigma(S\_ON\_pat) \\
&+ C \times \Sigma(S\_OP\_patb) \\
&+ \Sigma \frac{A(uR\_S\_ON\_patb)}{(uPCnt + B^{(uR\_S\_ON\_patb)}} \\
&- \Sigma PE[S\_ON\_patb] \\
&+ \Sigma \frac{A(uR\_S\_OP\_pat)}{(uPCnt + B^{(uR\_S\_OP\_pat)}} \\
&- \Sigma PE[S\_OP\_pat]
\end{aligned}
\tag{5-2}
$$

where $\Sigma PE[\ ]$ stands for the total number of PEs abandoned due to the S_OP_pat cells, or S_ON_patb cells, or whatever is contained within the bracket. This applies because, if two or more S_OP_pat cells or S_ON_patb cells are close on the same matrix row, they may both cause the same PE to be abandoned.

Values for three coefficients, A, B, and C, has to be determined. A decrease of the goodness function will reflect an increase in the total number of abandoned PEs, due to S_OP_pat cells or S_ON_patb cells, and the relationship is linear (two negative

terms in the function). The redundancy availability of the abandoned PEs reduces the seriousness of such a PE loss by contributing a certain amount of positive value into the goodness value. The question is how much should it compensate, in terms of the amount of redundancy and PE complexity? It is clear that, in any case, any occurrence of S_OP_pat and S_ON_patb will sacrifice PEs. This is not a desirable situation. Thus, in comparison with an assignment which does not sacrifice any PE, this should have a negative effect on the overall goodness value. However, the negative effect will be less if there is more redundancy available to compensate. The value of compensation should be less than the value of real loss. The remaining question now is how big this gap should be. The answer depends on the magnitude of the two coefficients A and B. The lifting factor A is set to reduce the difference between the pure, relative redundancy $\frac{uR\_S\_ON\_patb}{uPCnt}$, and its negative counterpart S_ON_patb, but not fully compensated. After many experiments, A was set equal to $\frac{(uPCnt + B)}{B}$, and B was chosen to lie somewhere between 1.05 – 1.1. In fact, the nearer is B to unity, the more linear is the increasing rate of the goodness function with an increase in redundancy, and the more loss is compensated. The difference between the negative contribution of S_OP_pat and S_ON_patb cells and the positive compensation of the associated redundancy is about $1 - \frac{1}{B}$ as a result of this setting of A and B. C is set to reduce the contribution of the S_OP_patb, and was chosen to be between 0.01 – 0.04. Actually, the choice of C is intuitive, because we have to consider its relation to the difference between the negative contribution of S_OP_pat and S_ON_patb cells and the associated compensation ($1 - \frac{1}{B}$, set earlier). The problem is that it is not clear by how many S_OP_patb cells the benefit brought should compensate this difference (the negative effect of sacrificing a PE). If 0.05 is chosen for C, then two S_OP_patb cells will compensate the negative effect caused by abandoning one PE and using one of its redundancy. After many experiments, a value of C = 0.02 appears an appropriate choice.

There are some situations where a net obviously can never be implemented in a matrix row. These include:

1. The number of PEs abandoned by S_ON_patb cells, created by an assignment, is more than the available redundancy for the PE. This means the number of PEs of that type required by the system specification are not available on the chip, causing immediate failure of the routing process.

2. The number of PEs abandoned by S_OP_pat cells, created by an assignment, is more than the total number of available PEs of the same type on the chip (the total number of unused PEs of the same type on the chip). This indicates that, if this assignment is chosen, the net can never be implemented in a matrix row, because not enough PEs of the same type can be found on the chip to make up the net. This causes immediate failure of the routing process. It can be noticed that we are comparing the number of abandoned PEs with the total number of available PEs of the same type on the chip, which is different from the S_ON_patb case. The reason for not comparing with the amount of redundancy is the fact that the S_OP_pat cells in this assignment do not necessarily inhibit the use of their related PEs entirely throughout the routing process. In other words, these PEs are only abandoned by this particular assignment. It is still possible for other nets on different matrix rows to use these PEs, without affecting the net creating these S_OP_pat cells due to open-circuits.

Considering all the different type of PEs, and writting these two situations in mathematical form, we have the conditions for inhibiting an assignment:

$$\Pi[PE[S\_ON\_patb] - (uR\_S\_ON\_patb)] > 0 \qquad (5\text{-}3)$$

and

$$\Pi[[PE[S\_OP\_pat] - (uR\_S\_OP\_pat)] - PE\_on\_Chip[S\_OP\_pat]] > 0 \qquad (5\text{-}4)$$

where PE[ ] indicates number of PEs being abandoned by S_ON_patb cells or S_OP_pat cells. PE_on_Chip[ ] indicates the number of unused PEs of the same kind abandoned by the S_OP_pat cell on the chip. Referring to all different kind of PEs in the chip, "AND" operation is carried out among them.

In the normal matrix architecture shown in Figure 2.6, all the PE's outputs are

hardwired to matrix rows. If a net contains the output port (node) of one kind of PE and there are i such kinds of PE in the chip, then only i matrix rows can be assigned to the net. All the other matrix rows should be given a special goodness value for this net to inhibit the assignment. For the duplex matrix architecture shown in Figure 2.8, all the matrix row can be assigned to any net. This is the difference in goodness function for the two different matrices.

5.4. The Algorithm

The process of system customisation (i.e. implementing all the required interconnections in the switch matrix) is carried out by mapping all the required PEs and their nets, as specified by system configuration information, one-by-one on to the superchip. If all the required PEs can be successfully mapped on to the superchip, then the customisation is successful, otherwise it has failed. For convenience in expressing the algorithm, the current PE being implemented is referred to as Cur(PE). The algorithm is presented as follows; each step is given in detail.

INPUT:

1. The size of the crossbar switch matrix (**n**, **m**).

2. The fault pattern indicating all the faulty switch cells and PEs, including their fault conditions (S_ON or S_OP for switch cells) and physical location. The data here are obtained directly from the test information for the switch matrix and the PEs on the chip.

3. Chip composition information: i.e. all the PEs available on the chip. This usually contains the locations of the different PEs, their names, the parameters for each PE, and some physical data, useful to the customisation such as the number of pins, their positions within the PE, their categories (input or output pin), etc.

4. System configuration information: i.e. the required interconnections for the system.

OUTPUT:

1. **Success.** All the desired system interconnections have been successfully embedded within the crossbar switch matrix and, therefore, the required system is correctly linked up. A list of data indicating the locations of those switch cells which should be turned on is given as an output.

2. **Failed.** At least one of the desired interconnections can not be embedded in the switch matrix. This may be caused by some faulty PEs damaged by processing defects or abandoned by some faulty switch cells. This results in the fact that there are not enough PEs to configure the system in this particular chip. This may happen quite early during the course of the routing (customisation). On the other hand, failure may result from the fact that there are too many faulty switch cells in the switch matrix and, more importantly, the distribution of these faulty cells is such that all the required interconnections for the system can not be made correctly in the switch matrix.

Overview of the Algorithm:

START

Step 1: Read all the information and put it into the proper data structures.

Step 2: Abandon the affected PEs.

Step 3: Build a priority queue for each net.

Step 4: Matrix row−net assignment.

Step 5: Get a PE for allocation.

Step 6: Find the port-related matrix rows and columns of the PE.

Step 7: Perform the mapping.

Step 8−10: Perform heuristic search for the best allocation of the PE.

END

Details of the Algorithm:

START

1. **Read All the Information and Put It Into the Proper Data Structures.** An integer array with the same size as the required crossbar switch matrix (n, m) is dynamically allocated for storing the information. While reading the fault pattern of the switch matrix, the information about the faulty switch cell is stored in this integer array according to location. The PE banks together with the information about their location are put on to a queue with links between the PE banks of the same type, so that these can easily be searched. Meanwhile, all the PEs specified in system configuration information together with their connection information are placed on another queue, **Q(PE)**, waiting for implementation.

2. **Abandon the Affected PEs.** This is the process whereby all the PEs on the chip are checked through their I/O pins. If there is any short among the I/O pins within a single PE by some switch failure in the matrix, for example, two neighbouring switch cells on the same matrix row are both S_ON, then this PE has to be abandoned. The other awkward situation occurs when at least one I/O pin of a PE is shorted to more than one matrix row. For example, more than one switch cell on the same matrix column is S_ON. This is a complicated situation. Clearly, there are two choices: abandon the PE, or abandon one or more shorted matrix rows. To make such a decision requires global system level consideration. However, the basic criterion should be based on the cost of such a decision. Such a cost depends upon two main factors: complexity and the availability of redundant parts (PE or matrix row). The availability of redundant parts usually has higher priority than complexity in the consideration, because this is directly related to the success of the whole routing (customisation) process. According to the redundancy selection strategy presented in Chapter 3, the PE is usually abandoned first whenever the redundancy availability is satisfied.

3. **Build a Priority Queue for Each Net.** First of all, a net list is generated from the information of the system configuration and interconnections. Then all the faulty switch cells on each matrix row, classified as S_ON_pat, S_ON_patb, S_OP_pat, and S_OP_patb, are counted to evaluate the goodness value for each net. If there are p nets and n matrix rows, there will be $p \times n$ such evaluations. All

the goodness values for one net will be put into a queue. Therefore, each queue will have n elements. To form a priority queue, all the goodness values in each queue are sorted by taking goodness value as a key. Since the goodness values are non-integer (i.e. reals), shell sort [131] is performed.

4. **Matrix Row-Net Assignment.** The assignment process tries to allocate a matrix row to a net. It scans all the net priority queues and finds the best assignments for all the nets by maximising the overall goodness value of the assignments. For a particular system configuration and a particular fault pattern of switch matrix, the overall goodness value is often spotted on a number of discrete values. It has been found during the experiments that two or more nets may have the same assignment to a particular matrix row, according to their goodness values in the priority queues. This means that assigning any of these nets to the matrix row will have the same overall goodness value. This introduces a conflict into the assignment of these nets. When a conflict occurs during the process of assignment, the matrix row will first arbitrarily be assigned to one of the conflicting nets. All the other conflicting nets will be recorded in case these assignments do not result in a successful configuration. Then other assignments with the same overall goodness value can be tried, by swapping the conflicting nets.

5. **Cur(PE) = the first element of the Q(PE).**

6. **Find the Port-Related Matrix Rows and Columns.** Once all the nets have been allocated to the matrix rows, a physical path for each net can be created by switching the correct switch cells on the matrix row, to which the net has been assigned. The activated switch cells will electrically connect the required I/O ports of the PEs together through the matrix columns. From the PE's point of view, a PE having i I/O ports should now have relations to i matrix rows. These rows are called port-related rows of the Cur(PE). The exact physical location where the switch cells should be activated must be determined by both the row number and the column number. For a particular net, the matrix row has been determined. However, finding the columns is more complicated, because the exact position of the Cur(PE) has not been decided yet. The relationship between the Cur(PE) and its physical image in the chip is not necessarily one-to-one. Instead, there may be several unused PEs which are the same as Cur(PE). To help map Cur(PE) on to a particular PE of its own kind, all these so far unused PEs will be considered as

possible candidates for the allocation. Thus each I/O port on the Cur(PE) may be related to several I/O ports (or matrix columns) on the superchip. The related matrix columns are called port-related columns of the Cur(PE). The step here is to establish the relationship between the Cur(PE) and all its possible candidates, including the information on its port-related rows and columns.

7. **Perform the Mapping.** This is the actual process of mapping Cur(PE) on to the superchip. Before mapping Cur(PE), every switch cell in the switch matrix and every I/O port of all the PEs on the chip are given a status, representing various states they might possibly have. For convenience in the mapping operation, each status is represented by some integers. Table 5.3 summarises each status used in the algorithm and its integer coding.

Table 5.3 Status Used in the Algorithm

Status	Meanings
0	working, unoccupied†
1	working, occupied
2	S_ON, unoccupied, unused (only for switch cell)
3	S_ON, occupied, unused (only for switch cell)
4	S_OP
5	not working (only for I/O ports of PEs)
6	special attention status††

† Occupied (unoccupied) means this switch cell or PE has (not) been allocated to a net to make a connection.

†† When a matrix column has been assigned to a port of the Cur(PE) during the mapping, any remaining S_ON switch cell in the matrix row related to this PE port (the matrix row assigned by the net of the PE port) is given this special attention status. During every assignment of a PE port, the whole column of that port is checked . If there is any "6" status in the column (except the point on the matrix row where the net is assigned), it indicates that this S_ON switch cell will short the port to some other net. To prevent this happening, the final mapping status should be a "5", which inhibits this PE as a candidate for the Cur(PE).

After establishing the status for all the points related to Cur(PE), the mapping operation is performed. This is achieved by first searching for any special attention status on the I/O port-related columns, but not on the I/O port-related rows, of the Cur(PE). Any such status will result in assigning a "5" status to all the I/O ports of the PE linked to the columns with the "6" status. Then, if the "6" status is found on an I/O port-related column as well as on an I/O port-related row, then the PE connected to this port-related column will have the highest priority chosen for the Cur(PE). The operations are performed between the status of all the port-related rows of the Cur(PE) and the status of the I/O ports of all the candidate PEs for the Cur(PE). All these status of I/O ports are overwritten by the mapping results. The next move is to choose which candidate should be allocated to the Cur(PE). To make this decision, the mapping operation is performed within the I/O ports of the each candidate, which results in k status values if there are k candidates in the superchip. Each of these final status values represents one of those candidates. The operational rules are given in Table 5.4. For the normal matrix architecture shown in Figure 2.6, the mapping is always performed amongst the matrix columns of all the input ports of the PEs, because all the output ports have been hardwired to the matrix rows.

Table 5.4 Operation Rules

	0	1	2	3	4	5
0	0	E†	2	E	5	5
1	E	E	E	1	E	5
2	2	E	2	E	E	5
3	E	1	E	E	E	5
4	5	E	E	E	E	5
5	5	5	5	5	5	5
6	6	1	6	1	6	5

† "E" is an error status. It means that the two operands should not meet each other during the mapping operation.

8. **if** there is any "6" in the final status,

then choose this candidate and set some occupation information about

the chosen PE

goto 10

else if there is any "2" in the final status,

then choose this candidate and set some occupation information about

the chosen PE

goto 10

else if there is any "0" in the final status,

then choose this candidate and set some occupation information about

the chosen PE

goto 10

else goto 9

endif

9. **if** there is at least one un-trialled conflict

in the record

then swap the conflicting nets,

set up a trialled status in the record (delete the record)

goto 5

else failed, **stop**.

endif

10. **if** it is not the end of the **Q(PE)**

then Cur(PE) = next PE in the **Q(PE)**

goto 6

else success, stop.

endif

end.

As the algorithm suggests, the search region for the best matrix row−net assignment is defined by all the recorded conflicting nets. There is a reason why only the conflicting nets are chosen to define this search region. Theoretically, if there exists a solution for the specified customisation within a particular superchip, then this solution can always be found by exhaustively searching all the possible combinations of matrix row−net assignments in the superchip. However, such exhaustive searching algorithm has been proved to be NP-Complete†. Therefore, it is not practical to perform exhaustive searching. In fact, the solution coverage of the search region defined in the algorithm is very high, because it is justified by the overall goodness value of the assignment. Such a goodness value is derived from the empirical consideration of all the global information to which the assignment can obtain maximum benefit.

5.5. Implementation

The above algorithm, called *router*, has been implemented using the C programming language in about 3,000 lines of code. To run this program, the user should provide four input files: the system configuration file, the chip composition file, and the files of the fault patterns of both the crossbar switch matrix and the PEs. Additionally, the size of the switch matrix (**n**, **m**) should be specified to the router. A warning message will be given if the given matrix size is too small to implement the required system.

† The formal mathematical proof of NP-Completeness can be found in Appendix B.

5.5.1. System Configuration File

Each statement in the system configuration file defines one PE required for the system. The statement should follow the syntax:

PE_Name Parameters SigIn SigOut CtrlIn CtrlOut

There is no restriction for the PE name; it can be letters, numbers or the combination of these two. The parameters introduced here are especially for bit-serial PEs, and contain the latency information. The same kind of PE may be required to have different latencies in different locations in a net. These are treated as different PEs by the program. All the rest is interconnection information, defining the connection of all the I/O ports of the PEs to the nodes in the system. The number of parameters, the number of signal input ports, signal output ports etc. are standard data and fixed for a particular PE. For example, a bit-serial adder should have four parameters, three signal inputs, two signal outputs, and one control input. This information is defined in a standard library. The router will consult the library during reading of the system configuration file, so that the router can correctly classify the data. The file should be named with "spec_d" in the current working directory. Figure 5.1 gives an example of such file.

5.5.2. Chip Composition File

Each statement in the chip composition file defines a group of PEs of the same kind (with the same parameters when they are bit-serial PEs) adjacent each other in the superchip. The syntax is

PE_Name Parameters Start_Column End_Column Up_Low Pin_Number Pin_category

"Start_Column", "End_Column" and "Up_Low" define the actual position of the PE bank in the superchip in which "Up_Low" refers to the position of being above or below the switch matrix. "1" indicates it is above the switch matrix and "0" indicates the other. "Pin_Number" gives the number of pins per each PE in the bank. Pin_category is a list of integers which gives the meaning of the pins for each PE, from the first pin on the left to the last pin on the right. To be consistent with the convention of the port status used in the program, 5, 9, 6, and 10 are chosen to represent SigIn, SigOut, CtrlIn, and CtrlOut, respectively. For example,

```
ADD          1  0  0  0  20  22  2  36  3  6
ADD          1  0  0  0  24  26  2  40  3  6
SUBTRACT     1  0  0  0  20  22  2  37  3  6
SUBTRACT     1  0  0  0  24  26  2  41  3  6
ADD          1  1  1  0  21  23  2  38  3  7
ADD          1  1  1  0  25  27  2  42  3  7
SUBTRACT     1  1  1  0  21  23  2  39  3  7
SUBTRACT     1  1  1  0  25  27  2  43  3  7
ADD          1  1  0  0  36  38  2  28  3  8
ADD          1  1  0  0  40  42  2  32  3  8
SUBTRACT     1  1  0  0  36  38  2  30  3  8
SUBTRACT     1  1  0  0  40  42  2  34  3  8
ADD          1  1  0  0  37  43  2  29  3  9
ADD          1  1  0  0  41  39  2  35  3  9
SUBTRACT     1  1  0  0  37  43  2  31  3  9
SUBTRACT     1  1  0  0  41  39  2  33  3  9
CBITDELAY    1  5  6
CBITDELAY    1  5  7
CBITDELAY    1  6  8
CBITDELAY    1  7  9
```

Figure 5.1 Example of a system configuration file

ABSOLUTE 14 0 48 124 0 2 5 9

gives a PE bank called **ABSOLUTE 14 0**. It starts from the matrix column 48 and ends at the column 124 with 2 I/O ports for each element (that means there are 38 such identical PEs in the bank). The first I/O port is a signal input and the second one is a signal output. The name of the composition file should be called "loc_d" in the current working directory.

5.5.3. Fault Pattern Information

Fault patterns are formed from testing the superchip. There are actually two fault pattern files: one for the switch matrix and one for PE banks. They are called "testsm_d" and "testp_d", respectively, in the current working directory. The syntax for the switch matrix fault pattern is

Failure_Type Row_Number Column_Number

where "Failure_Type" usually is an integer, "2" for S_ON and "4" for S_OP. "Row_Number" and "Column_Number" define the location of the fault in the switch matrix. To be consistent with the convention in C language, "Row_Number" and "Column_Number" start from 0 for the first matrix row or column. The syntax for the fault pattern of the PE banks is

PE_Name Start_Column Pin_Number Up_Low

This defines the location of one faulty PE in the superchip. If there are two or more faulty PEs in the same bank, they have to be listed with two or more such statements.

5.6. An Example†

In this section, the message routing for a DFT chip is taken as an example of using the routing algorithm to customise a superchip. Figure 5.2 shows the organisation of the superchip for the DFT implementation without redundancy. The crossbar switch matrix in the middle has a size of 31×104. The data given in Figure 5.1 are the system configuration file. Figure 5.3 is the chip composition file. Several different fault

† The DFT architecture in this example is taken from a study given in [132].

patterns were taken to see if the algorithm is efficient enough to avoid and/or make use of the faulty switch cells, as much as possible. Figure 5.4 gives a fault pattern for the switch matrix. There is no critical fault in the PE bank. Figure 5.5 shows the routing output of the system under the fault pattern in the switch matrix. In the output diagram, S_OP switch cell is indicated by a red box and S_ON switch cell is indicated by a green box. Increasing the number of faults in the switch matrix as shown in Figure 5.6, the routing result is changed and shown by Figure 5.7. Comparing Figure 5.5 and 5.7, the results indicate that, because of its unique architecture, the crossbar switch matrix can tolerate lots of faults in the matrix itself even without any redundancy.

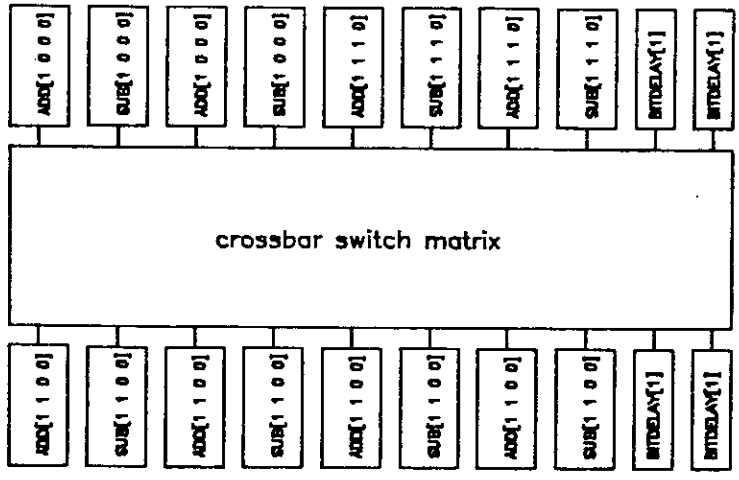

Figure 5.2 The system schematic for a DFT

When the number of faulty switch cells in the switch matrix is further increased, the resulting large number of faults in the switch matrix may require several PEs to be abandoned, depending on where the faults are. Figures 5.8 and 5.9 are two outputs of denser fault patterns with two redundancies **ADD[1 1 0 0]** and **SUBTRACT[1 1 0 0]** as listed at the bottom of Figure 5.3. Here, all the faults are artificially concentrated in the switch matrix rather that having some in the PEs as a random distribution. The purpose of introducing this example is to demonstrate the effectiveness of the routing algorithm at tolerating faults within the switch matrix. This is more complicated than tolerating faulty PEs. In fact, tolerating faulty PEs is very straightforward as far as the algorithm is concerned. Figure 5.10 shows the actual output file from the *router* for the output depicted in Figure 5.9. In part of "OUTPUT ROUTING DATA", the number of brackets equals the number of matrix columns starting from the first column on the left to the last column on the right. The number in the brackets shows that the switch cell

```
ADD      1  0  0  0  0  5   1  6  5  5  5  9  9  6
ADD      1  1  0  0  6  11  0  6  5  5  5  9  9  6
SUBTRACT 1  0  0  0  12 17  1  6  5  5  5  9  9  6
SUBTRACT 1  1  0  0  18 23  0  6  5  5  5  9  9  6
ADD      1  0  0  0  24 29  1  6  5  5  5  9  9  6
ADD      1  1  0  0  30 35  0  6  5  5  5  9  9  6
SUBTRACT 1  0  0  0  36 41  1  6  5  5  5  9  9  6
SUBTRACT 1  1  0  0  42 47  0  6  5  5  5  9  9  6
ADD      1  1  1  0  48 53  1  6  5  5  5  9  9  6
ADD      1  1  0  0  54 59  0  6  5  5  5  9  9  6
SUBTRACT 1  1  1  0  60 65  1  6  5  5  5  9  9  6
SUBTRACT 1  1  0  0  66 71  0  6  5  5  5  9  9  6
ADD      1  1  1  0  72 77  1  6  5  5  5  9  9  6
ADD      1  1  0  0  78 83  0  6  5  5  5  9  9  6
SUBTRACT 1  1  1  0  84 89  1  6  5  5  5  9  9  6
SUBTRACT 1  1  0  0  90 95  0  6  5  5  5  9  9  6
CBITDELAY   1  96  99  1  2  6  10
CBITDELAY   1  100 103 0  2  6  10

!  the following two CEs are added with two redundancies for
!  Figure 9 and Figure 10.

ADD      1  1  0  0  104 109 1  6  5  5  5  9  9  6
SUBTRACT 1  1  0  0  110 115 0  6  5  5  5  9  9  6
```

Figure 5.3 The chip composition file for the DFT system

2	30	4
2	6	13
2	10	51
2	30	82
2	18	63
4	10	2
4	22	51
4	20	20
4	18	7
4	24	101

Figure 5.4 A fault pattern for the switch matrix

2	30	4
2	6	13
2	10	51
2	30	82
2	18	63
4	10	2
4	22	51
4	20	20
4	18	7
4	24	101
4	21	63
2	13	56
2	0	24
2	20	96
4	9	20
4	4	27
4	23	61

Figure 5.6 A second fault pattern for the switch matrix with more faults

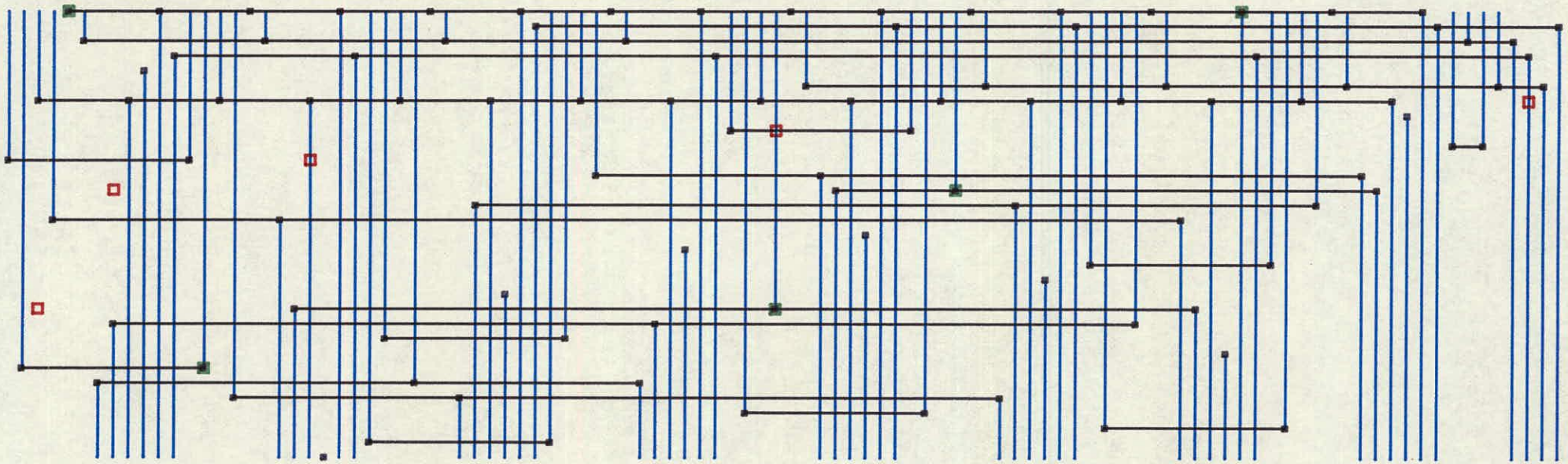

Figure 5.5 Routing output with the fault pattern given in Figure 5.4

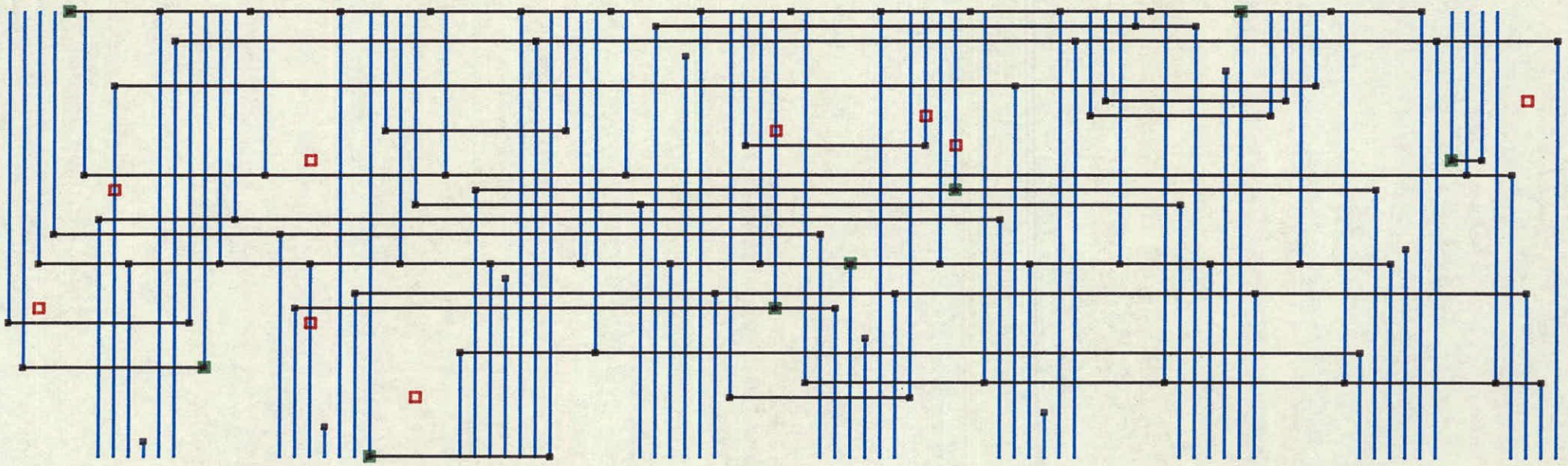

Figure 5.7 Routing output with the fault pattern given in Figure 5.6

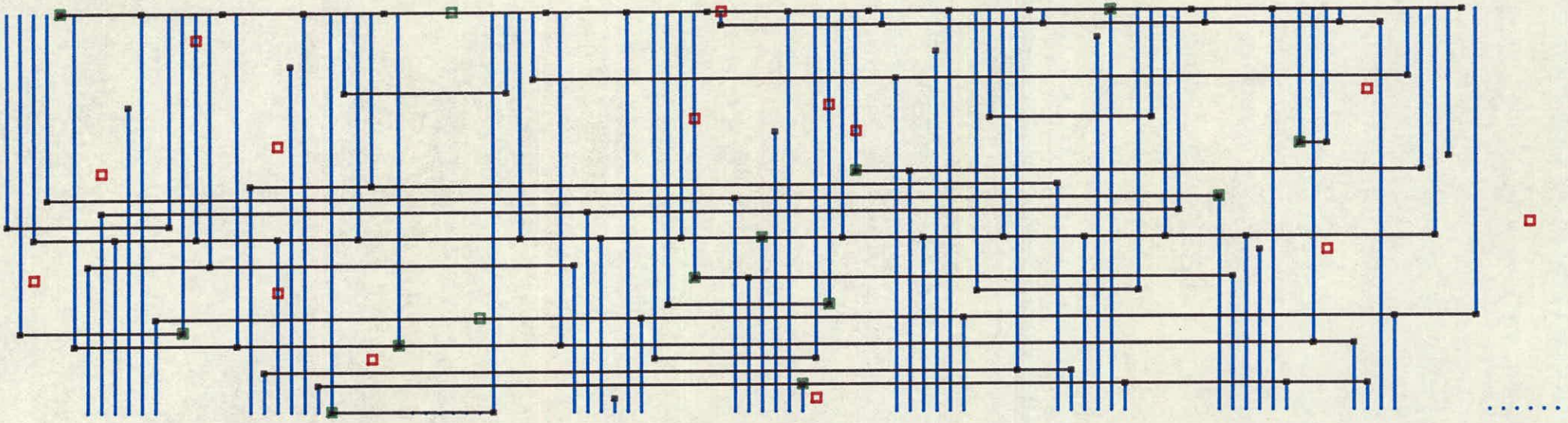

Figure 5.8 Output from routing (one redundancy is used).

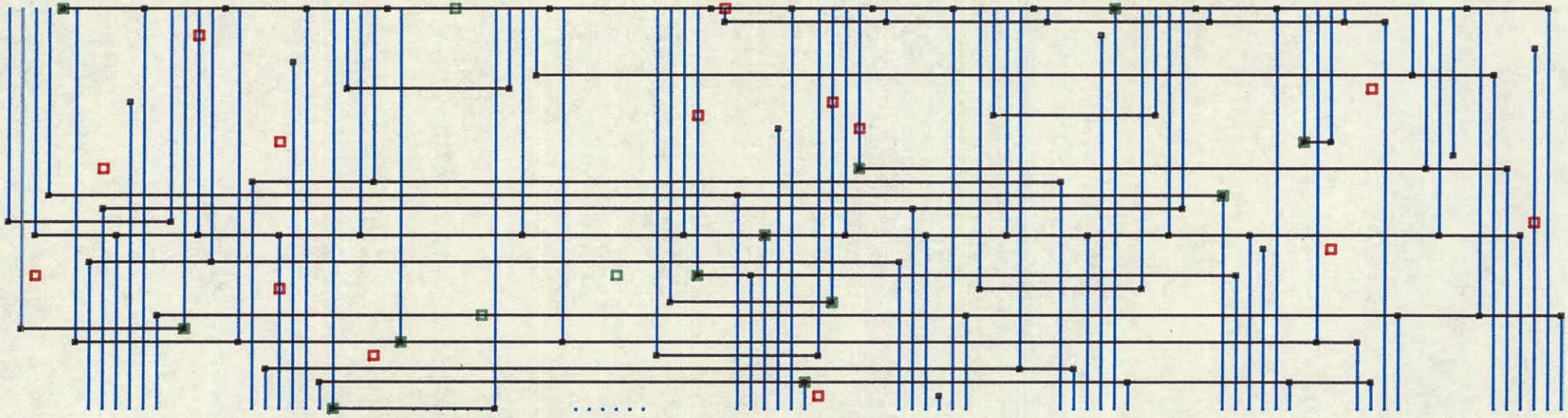

Figure 5.9 Output from routing (two redundancies are used).

```
Routing Output Data Generated by ROUT [Version 1.1]
        at:     14:40:37, on Wed May 21  1986

READ-IN SWITCH MATRIX TEST DATA:

TSC 2, 30, 4;
TSC 2, 6, 13;
TSC 2, 10, 51;
TSC 2, 30, 82;
TSC 2, 18, 63;
TSC 4, 10, 2;
TSC 4, 22, 51;
TSC 4, 20, 20;
TSC 4, 18, 7;
TSC 4, 24, 101;
TSC 4, 21, 63;
TSC 2, 13, 56;
TSC 2, 0, 24;
TSC 2, 20, 96;
TSC 4, 9, 20;
TSC 4, 4, 27;
TSC 4, 23, 61;
TSC 2, 30, 33;
TSC 2, 10, 45;
TSC 4, 28, 14;
TSC 4, 1, 60;
TSC 4, 12, 98;
TSC 4, 30, 53;
TSC 2, 16, 90;
TSC 2, 2, 59;
TSC 2, 5, 29;
TSC 2, 8, 61;
TSC 4, 14, 113;
TSC 2, 7, 35;
READ-IN PRIMITIVE TEST DATA:

(Primitive Banks Perfect)

READ-IN PRIMITIVE POSITION DATA FOR THE CHIP:

PP ADD[1,0,0,0], 0, 5, 6, 1;
PP ADD[1,1,0,0], 6, 11, 6, 0;
PP SUBTRACT[1,0,0,0], 12, 17, 6, 1;
PP SUBTRACT[1,1,0,0], 18, 23, 6, 0;
PP ADD[1,0,0,0], 24, 29, 6, 1;
PP ADD[1,1,0,0], 30, 35, 6, 0;
PP SUBTRACT[1,0,0,0], 36, 41, 6, 1;
PP SUBTRACT[1,1,0,0], 42, 47, 6, 0;
PP ADD[1,1,1,0], 48, 53, 6, 1;
PP ADD[1,1,0,0], 54, 59, 6, 0;
PP SUBTRACT[1,1,1,0], 60, 65, 6, 1;
PP SUBTRACT[1,1,0,0], 66, 71, 6, 0;
PP ADD[1,1,1,0], 72, 77, 6, 1;
PP ADD[1,1,0,0], 78, 83, 6, 0;
PP SUBTRACT[1,1,1,0], 84, 89, 6, 1;
PP SUBTRACT[1,1,0,0], 90, 95, 6, 0;
PP CBITDELAY[1], 96, 99, 2, 1;
PP CBITDELAY[1], 100, 103, 2, 0;
PP ADD[1,1,0,0], 104, 109, 6, 1;
PP SUBTRACT[1,1,0,0], 110, 115, 6, 0;


DIFFERENT PRIMITIVES MET IN THE SPECIFICATION:

ADD[1,0,0,0]
SUBTRACT[1,0,0,0]
ADD[1,1,1,0]
SUBTRACT[1,1,1,0]
ADD[1,1,0,0]
SUBTRACT[1,1,0,0]
CBITDELAY[1]

NET-PRIMITIVE LIST:


**** Net1 ****
SUBTRACT[1,1,0,0],
SUBTRACT[1,1,0,0],
ADD[1,1,0,0],
ADD[1,1,0,0],
SUBTRACT[1,1,0,0],
SUBTRACT[1,1,0,0],
ADD[1,1,0,0],
ADD[1,1,0,0],
SUBTRACT[1,1,1,0],
SUBTRACT[1,1,1,0],
ADD[1,1,1,0],
ADD[1,1,1,0],
SUBTRACT[1,0,0,0],
SUBTRACT[1,0,0,0],
ADD[1,0,0,0],
ADD[1,0,0,0],

**** Net2 ****
SUBTRACT[1,1,0,0],
SUBTRACT[1,1,0,0],
ADD[1,1,0,0],
ADD[1,1,0,0],
SUBTRACT[1,1,0,0],
SUBTRACT[1,1,0,0],
ADD[1,1,0,0],
ADD[1,1,0,0],
SUBTRACT[1,1,1,0],
SUBTRACT[1,1,1,0],
ADD[1,1,1,0],
ADD[1,1,1,0],
SUBTRACT[1,0,0,0],
SUBTRACT[1,0,0,0],
ADD[1,0,0,0],
ADD[1,0,0,0],

**** Net19 ****
SUBTRACT[1,0,0,0],
ADD[1,0,0,0],

**** Net20 ****
SUBTRACT[1,1,1,0],
ADD[1,1,1,0],

**** Net21 ****
SUBTRACT[1,0,0,0],
ADD[1,0,0,0],

**** Net22 ****
SUBTRACT[1,1,1,0],
ADD[1,1,1,0],

**** Net23 ****
SUBTRACT[1,0,0,0],
ADD[1,0,0,0],

**** Net24 ****
SUBTRACT[1,1,1,0],
ADD[1,1,1,0],

**** Net25 ****
SUBTRACT[1,0,0,0],
ADD[1,0,0,0],

**** Net26 ****
SUBTRACT[1,1,1,0],
ADD[1,1,1,0],
```

Figure 5.10 The actual output file

```
**** Net27 ****
ADD[1,1,0,0],

**** Net28 ****
ADD[1,1,0,0],

**** Net29 ****
SUBTRACT[1,1,0,0],

**** Net30 ****
SUBTRACT[1,1,0,0],

**** Net31 ****
ADD[1,1,0,0],

**** Net32 ****
SUBTRACT[1,1,0,0],

**** Net33 ****
SUBTRACT[1,1,0,0],

**** Net34 ****
ADD[1,1,0,0],

**** Net35 ****
SUBTRACT[1,1,0,0],
ADD[1,1,0,0],
ADD[1,0,0,0],

**** Net36 ****
SUBTRACT[1,1,0,0],
ADD[1,1,0,0],
SUBTRACT[1,0,0,0],

**** Net37 ****
SUBTRACT[1,1,0,0],
ADD[1,1,0,0],
ADD[1,1,1,0],

**** Net38 ****
SUBTRACT[1,1,0,0],
ADD[1,1,0,0],
SUBTRACT[1,1,1,0],

**** Net39 ****
SUBTRACT[1,1,0,0],
ADD[1,1,0,0],
ADD[1,0,0,0],

**** Net40 ****
SUBTRACT[1,1,0,0],
ADD[1,1,0,0],
SUBTRACT[1,0,0,0],

**** Net41 ****
SUBTRACT[1,1,0,0],
ADD[1,1,0,0],
ADD[1,1,1,0],

**** Net42 ****
SUBTRACT[1,1,0,0],
ADD[1,1,0,0],
SUBTRACT[1,1,1,0],

**** Net44 ****
CBITDELAY[1],
CBITDELAY[1],

**** Net45 ****
CBITDELAY[1],
CBITDELAY[1],
SUBTRACT[1,0,0,0],
SUBTRACT[1,0,0,0],
ADD[1,0,0,0],
ADD[1,0,0,0],

**** Net46 ****
CBITDELAY[1],
CBITDELAY[1],
SUBTRACT[1,1,1,0],
SUBTRACT[1,1,1,0],
ADD[1,1,1,0],
ADD[1,1,1,0],

**** Net47 ****
CBITDELAY[1],
SUBTRACT[1,1,0,0],
SUBTRACT[1,1,0,0],
ADD[1,1,0,0],
ADD[1,1,0,0],

**** Net48 ****
CBITDELAY[1],
SUBTRACT[1,1,0,0],
SUBTRACT[1,1,0,0],
ADD[1,1,0,0],
ADD[1,1,0,0],


SOME STATISTICS:

Number of Signal Nets:                  43
Number of Control Nets:                      9
Total Number of Nets:                   52
Number of Primitives read:              20
Number of Signal Ports:                      80
Number of Control Ports:                24
Total Number of Ports:                  105
Number of Primitive Banks:              20
Number of S_ON cells:                   15
Number of S_OP cells:                   14
Number of Faulty Primitives             0
Number of Used Special Atten. Cells     6
Number of s_on used for routing:        6
Rerouting times:                        2
cpu time (User):                        28.5 [sec]
cpu time (System):                      6.6 [sec]

OUTPUT ROUTING DATA:

(14) (6) (13) (16) (30) (5) (11) (15) (13) (23) (30) (7) (14) (6)
(13) (11) (30) (5) (17) (3) (13) (26) (30) (2) (0) (24) (13) (17)
(30) (5) (NC) (NC) (NC) (NC) (NC) (NC) (0) (24) (13) (25) (30) (5)
(NC) (NC) (NC) (NC) (NC) (NC) (4) (8) (13) (10) (30) (29) (16) (10)
(13) (21) (30) (2) (4) (8) (13) (18) (30) (29) (11) (15) (13) (1)
(30) (7) (9) (22) (13) (3) (30) (29) (17) (3) (13) (28) (30) (2) (9)
(22) (13) (15) (30) (29) (16) (10) (13) (12) (30) (2) (20) (5) (20)
(29) (5) (2) (29) (7) (25) (18) (13) (19) (30) (7) (25) (18) (13)
(27) (30) (7)

route: The whole routing finished successfully!
```

Figure 5.10 The actual output file (continued)

on that row of the column should be switched to the electrically conducting state. "NC" means "no connection" for that column.

5.7. Discussion

As shown in the example above, the success of system routing does not only depend upon the fault density of the chip (the number of faults), but also upon the actual fault pattern (fault distribution) as well. However, the yield model developed earlier in Chapter 3 was solely based on defect density (or fault density). The fault pattern sensitivity of the algorithm shown in the experiments here suggests a vital need to explain the yield model in predicting the superchip yield. Actually, there is no contradiction between fault pattern sensitivity and the yield model, based solely on fault density. The yield model gives a statistical average success of message routing, for a given fault level existing on the chip. A different distribution of these faults may cause some variation around the average figure.

The effect of critical defect clustering in the crossbar switch matrix has been studied during the routing experiments. With larger defect clusters in the switch matrix, it is more likely that one or two working PEs are abandoned. If there is PE redundancy, the whole routing will consequently be quite similar to the routing with nearly perfect elements on the chip (because more clustering at one place means less faulty elements on the rest of the chip). Therefore, the possibility of getting successful routing becomes greater. If there is not enough redundancy for the PE, this clustering could be disastrous to routing. Thus, going back to redundancy selection, it should be ensured that enough redundancy is available in the sense of tolerating fault clusters. This applies particularly for PEs placed near the edge of the wafer (in which case some additional redundancy should be added).

5.8. Conclusion

An automatic routing algorithm for a partially defective crossbar switch matrix and its system realisation have been described in this Chapter. Various experiments have been carried out to test the performance of the algorithm. The problem is NP-Complete. However, the heuristic searching derived from the goodness function seems to be satisfactory. For a system with about 100 inter-PE connections, the *router* usually takes about 25 seconds on a VAX-750 to search for a solution, in the case where there

exist faults in the matrix. If the matrix is perfect, the searching time will be much less. The algorithm is not necessarily restricted to bit-serial systems as in the example given.

CHAPTER 6

A COLUMN FFT ENGINE USING THE CROSSBAR SWITCH MATRIX — A CASE STUDY

A case study of a flexible, high performance FFT engine, realised with the superchip architecture introduced earlier, is presented in this Chapter. This example has been chosen to show how the superchip architecture can be tailored to carry out a specific system computation.

6.1. Fast Fourier Transform

The computation of the Discrete Fourier Transform (DFT) is of considerable importance in digital signal processing applications. In such applications, transformation speed is one of the most important factors and has, hence, attracted much attention. Research has resulted in various algorithms for speeding up the transformation, notably the Fast Fourier Transform (FFT). This algorithm has long been one of the major analytical tools in the digital signal processing field for system analysis and power spectrum analysis, etc. In the early stages of FFT development, the main research effort was concentrated on the software realisation of the algorithm with the then available general purpose computers. Thus the speed of such FFT transformation was largely dependent upon the computer employed and the effectiveness of the mapping from the algorithm to the actual software realisation. With the advent of VLSI, the processing elements become ever cheaper, so that it is possible to realise the whole FFT in a specially designed hardware system, which can be much faster and more efficient than its software equivalent. Furthermore, due to its architectural modularity, the FFT is very suitable for VLSI implementation.

Certain single-chip Fourier transform processors have been published [133,134,135] recently. However, these chips are designed for specific transform lengths. The FFT engine discussed here is capable of carrying out Fourier transforms

with various lengths. Bit-serial architecture has also been adopted for the FFT engine for its simple communications and control, efficient computation, and better $area \times time^2$ performance. Figure 6.1 shows a 64 point, radix-4, FFT flow diagram taken from [136]. Each black point in the diagram represents a sample datum and each hole represents a radix-4 butterfly processor, with the usual 'twiddle' factors being indicated by arrows. Each radix-4 butterfly performs three complex multiplications and eight complex additions. The block diagram of a radix-4 butterfly processor is shown in Figure 6.2. To transform 64 samples from the time to the frequency domain, these samples have to be processed through three columns (stages) of the butterfly operations. According to the flow diagram, there are 16 butterfly processors in a column for a 64 point transform. The algorithm employed in Figure 6.1 is called the radix-4, normal-order input, digit-reversed output, in-place algorithm. The concept of column FFT processing involves trying to use repeatedly one column of the butterfly processors to carry out the whole transform. For example, in the flow diagram of Figure 6.1, a butterfly processor column is used to process the first column of the transform, then the outputs are properly re-ordered and presented again to the butterfly processor column to process the second column of the transform, and so forth. The so-called "Constant Geometry Algorithm" [136] has been identified in this research as being particularly suited for the column FFT process. Unlike the algorithm represented in Figure 6.1, where the topologies for transferring output data after each different column processing are different, the constant geometry algorithm provides a constant topology for transferring the output data after each column process. Consequently, a column of butterfly processors can be used repeatedly, without changing any of its structure during the course of the whole transformation. The constant geometry algorithm is achieved by including a 'perfect shuffle'† after each column. Figure 6.3 shows a 64-point, radix 4, constant geometry FFT column with the perfect shuffle. The FFT algorithms shown in both Figure 6.1 and 6.3 will be used as examples in the case study in this Chapter.

† The concept of the "perfect shuffle" and its precise definition can be found on pp. 210 in [137], or [138].

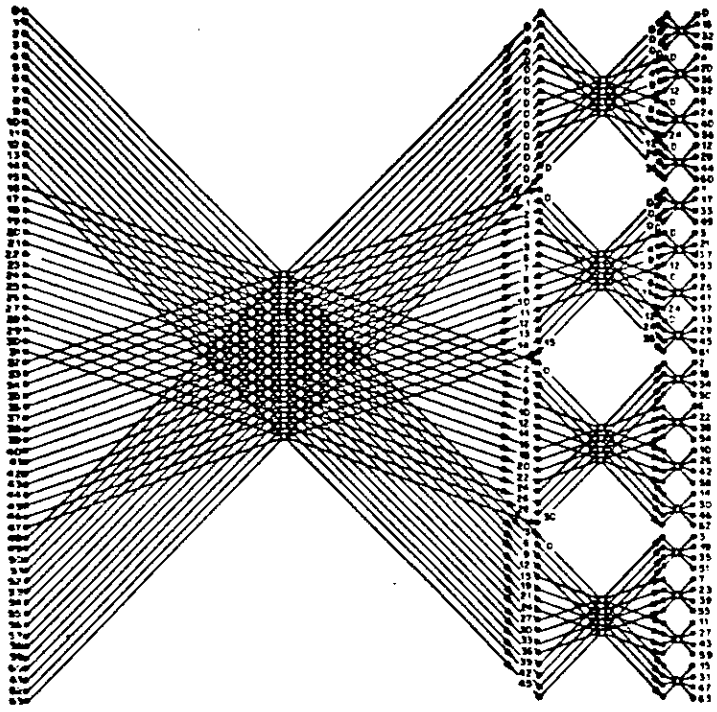

Figure 6.1 A 64-point, radix-4, DIF, FFT, normally ordered input, digital reversed output.

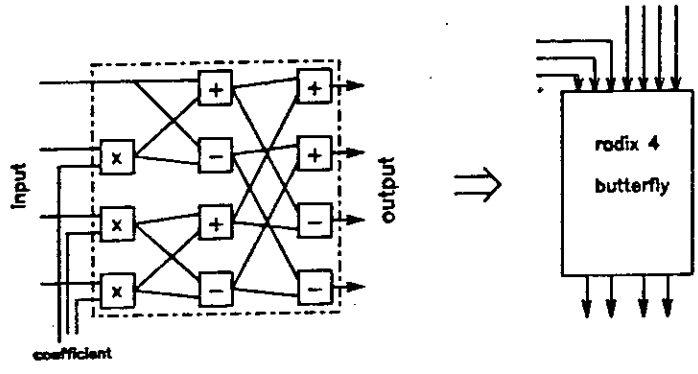

Figure 6.2 A radix-4 butterfly processor

6.2. A Monolithic 64 Point Column FFT System

6.2.1. System Organisation

The functional part of a 64-point column FFT system consists of 16 radix-4 butterfly processors. Each butterfly processor performs three complex multiplications and eight complex additions. Each input and output of a butterfly processor goes through a FIFO (First-In First-Out) register. They are called *input FIFOs*, and *output FIFOs*,

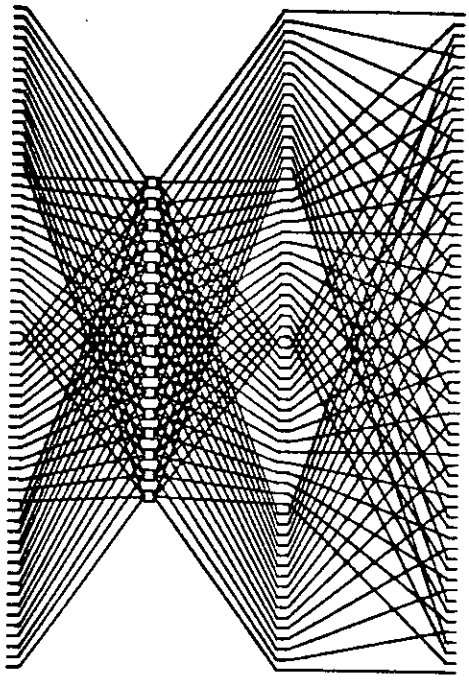

Figure 6.3 A stage of a 64-point constant geometry algorithm

respectively, as shown in Figure 6.4. The length of the FIFOs can be programmed to align the signals throughout the operation. This is particularly necessary for a bit-serial processing system, in which the wordlength often grows after certain arithmetic operations like addition. Furthermore, the system can also be easily geared to various wordlengths in different applications. An efficient variable-length shift register design [139] has been proposed. This shift register structure is very suitable for our purpose here, because the required change in delay is quite small, and the amount of delay does not need to be set to several different values, as required by some other variable-length shift registers [140]. A butterfly processor, and the eight associated FIFOs, form a complete processing element referred to here as the *butterfly processing element*, or BPE for short.

Pursuing the superchip architecture, all the BPEs are placed above and below a crossbar switch matrix as shown in Figure 6.5, in which the duplex matrix architecture is used. Every FIFO is connected to a column of the crossbar switch matrix, so that BPEs can only communicate with each other through its own FIFO, and the switch matrix. The crossbar switch matrix itself contains control circuits for system message routing. To be easily accessible from external signals, each matrix row is connected to a bidirectional I/O device shown in Figure 6.6. This bidirectional I/O device consists of

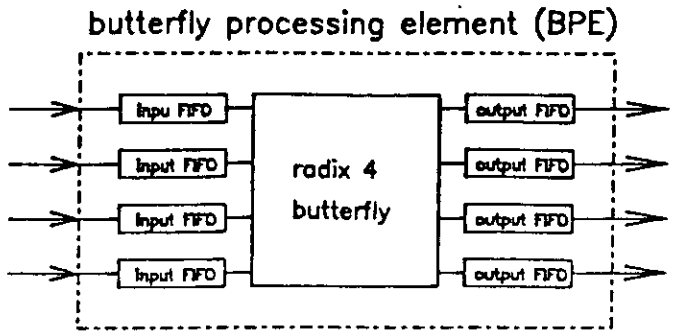

Figure 6.4 A radix-4 butterfly processing element

two, tri-state, non-inverting buffers. The "RCtrl" (Row Control) signal can disconnect a matrix row from externals.

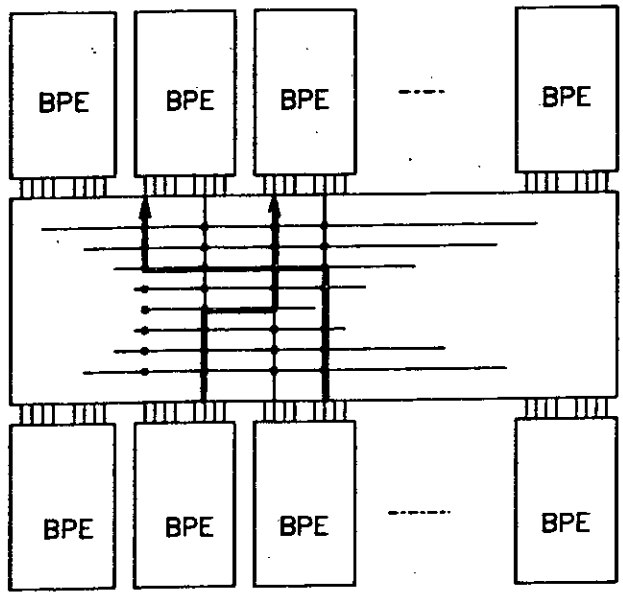

Figure 6.5 A schematic of the column FFT organisation

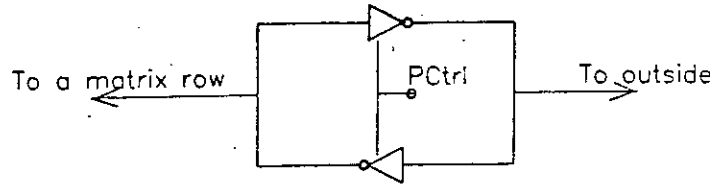

Figure 6.6 Bidirectional I/O device

For a 64-point transform length with complex data, the number of rows in the switch matrix should be 128, the number of columns should be 256.

6.2.2. System Operation

Figure 6.5 shows only a schematic of the system with an indication of data flow. Control in a bit-serial system involves control of the timing and the flow of events. The convention [88] for describing control in bit-serial systems is adopted here; it has already been presented in Section 2.4.3 and depicted in Figure 2.18. In the FFT system studied here, the control signals **c0**, **c1**, and **c2** are assumed to be automatically available from an off-chip clock generator.

Before performing the transformation, all the 64 samples are loaded into their corresponding input FIFOs through the bidirectional I/O devices and the switch matrix. Thereafter, the matrix rows are disconnected from externals by setting the "RCtrl" signals of the I/O devices. Then, the whole switch matrix is initialised by setting both row and column decoders of the matrix to their "clearing state". Immediately after initialisation, all the samples are fed into the butterfly processors in their own BPEs to start the first column computation. At the same time, message routing is carried out within the matrix for transferring output data from the current column processing. For 64-point column processing, there should be 16 BPEs for each column, each of which has been defined in the chip during the generation of the routing message, discussed in Chapter 5. This corresponding relationship does not change during the column processing. To have a correct transform result, message routing must finish before the output data for transfer appear at the matrix. By disregarding output FIFOs, the time permitted to complete message routing is the latency of the radix-4 butterfly processor. This latency depends on the number of bits of the coefficients for the complex multiplication. If the latency is less than the time that the message routing process requires, an extra amount of delay for the output data from the butterfly processors must be introduced through the output FIFOs to ensure that the output data appear after the message routing has been finished. The condition that the whole message routing finishes BEFORE the output data from the butterfly processors appear is called a *timing condition*. The butterfly computations and the system message routing set-up are two parallel processes executed at different parts of the superchip. The timing condition indicates that the time taken by the message routing process must not be more than that for the computation process.

The employment of the input and output FIFOs in each BPE has made it possible to keep an accurate timing control of these two parallel processes. Ultimately, the time period of these two parallel processes should be adjusted to be "just" the same so that the maximum throughput can be achieved. In other words, the time for the message routing should be adjusted to be the same as that for the butterfly computation. The condition for achieving this maximum throughput is called the *ultimate timing condition*.

When the routing time in the switch matrix is considerably longer than the latency of a BPE, a considerable proportion of the time for processing one column will not be spent on computation. Instead, it is spent waiting for the completion of the message routing process. To speed up the message routing process in this case, a so-called "parallel loading scheme" can be performed. Figure 6.7 shows parallel loading which is capable of embedding two routes at a time. In this scheme, the switch matrix is divided into two sections. There are two sets of addressing circuits which provide for these two sections. Two different routing messages are fed to these two addressing circuits in parallel. As a result, the message routing time is halved for a minor hardware overhead (one column decoder). The message routing time can be further reduced by dividing the switch matrix, and by adding more addressing circuits to address each small region of the switch matrix in parallel, so that the ultimate timing condition is always assumed to apply in the rest of the discussion.

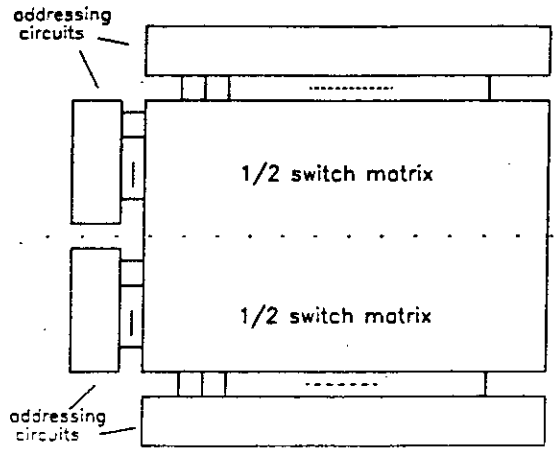

Figure 6.7 Parallel loading establishing two routs at a time

In the constant geometry algorithm, the system message routing is only carried out once during the processing of the first column. For the two remaining columns, the

topology of the data transformation does not change. The data can be sent directly into the switch matrix without any delay through the output FIFOs. The amount of delay set for the input FIFOs of the BPE of the next column, in this case, depends on the system wordlength and the latency of the butterfly processor. If the latency is longer than the system wordlength, then the delays of the input FIFOs can be set to zero. This follows because when the data processed from a current column are ready to be fed back to the next column, the whole word of the input data for the current column has already been injected into its butterfly processor, so that the coming data will not corrupt the previous data at their destinations. If the system wordlength is longer than the latency, then the delay of the input FIFOs are set to compensate for the difference so that next column processing starts after the whole word of all current column inputs has been injected into the butterfly processor. In the former situation, the computation is not efficient enough because part of each butterfly processor is idle all the time. Suppose the butterfly processor works in a pipeline fashion in term of the bits of the data it processes. Figure 6.8 (a), for example, shows the position of a word relating to a butterfly at the very beginning, t_o, when the first column is about to start. The length of the butterfly processor in the diagram represents its latency, and the length of the data represents wordlength. Figure 6.8 (b) shows the position after a time period of the butterfly processor latency, t_L, which indicates that part (front) of the processor is idle. After this point, any bit output from the first column are fed back immediately to their corresponding butterfly processors for the second column processing. Figure 6.8 (c) illustrates the position after the time $t_o + t_L$, and Figure 6.8 (d) shows the position when the first column processing is completed, both of which indicate part of the butterfly processor is idle throughout the processing. The ideal situation is to choose the number of bits for system wordlength such that the latency for the butterfly processor is the same as the system wordlength, thus eliminating idle periods. This situation is assumed in all remaining discussion unless otherwise stated. Figure 6.9 shows the timing control of these two overlapping processes. To distinguish from another column operation discussed later, this column operation is called *full column processing*. In full column processing, all output data have their own transferring path to their destinations after every column computation, i.e. the whole transform length can be processed and transferred concurrently, without any conflicting traffic. In Figure 6.9, the last message routing in the switch matrix, parallel to the last column computation, is to re-order the output into natural order.

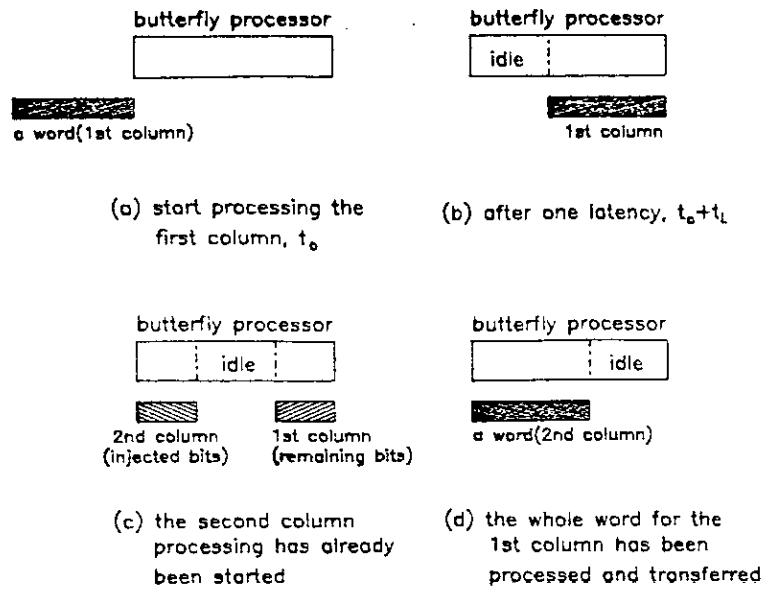

Figure 6.8 Relationship between latency and system wordlength (length of the butterfly processor represents the latency).

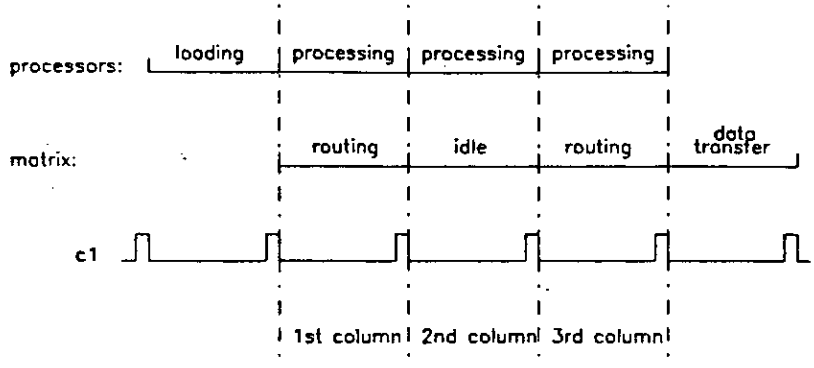

Figure 6.9 Timing overlap of full column processing

6.2.3. Chip Complexity

A bit-serial multiplier is already under development by other researchers [141,142], within the Department of Electrical Engineering, using 3μm CMOS technology in gate matrix style. This design is taken here as an example for analysing the complexity of the column FFT system. The result of this multiplier design suggests that the transistor count for one slice of the multiplier (one such slice mainly performs one bit multiplication for the coefficient) is around 200 and the silicon area for the slice is around $180\times1560\ \mu m^2$. A radix-4 butterfly processor mainly consists of three complex multipliers and some adders and subtractors. Suppose that the coefficient for the

multiplication is 12 bits in length. This will make a real multiplier use 12 slices and a radix-4 butterfly processor will use 144 slices, with a transistor count of around 29,000. Considering the control circuitry, input and output FIFOs, and some adders and subtractors required for the BPE, the total transistor count will be about 30,000. Each BPE will have 16 I/O ports communicating through the switch matrix. If the BPEs are placed both above and below the switch matrix, according to the width of the switch cell (132 μm), the pitch for each BPE will be about 4.2 mm. Therefore a BPE with 30,000 transistors will probably have a height of around 13mm. A switch matrix with 128 rows and 256 columns will occupy a silicon area of $7.2\times34\ mm^2$. Also including some I/O pads (64 bidirectional I/O pads for data plus some control and power pads), the size of the whole chip is around $35\times35\ mm^2$. This information is summarized in Table 6.1.

Table 6.1 Summary of the complexity of different components

Components	Transistor count*	Estimated area (mm^2)
One bit slice	200	1.5×0.18
Real multiplier	2,400	1.5×2.2
Complex multiplier	9,600	3.2×13.0
One BPE	30,000	4.2×13.0
switch matrix	34,000	34×7.2
64 point FFT	520,000	35×35

6.2.4. Yield Estimation

The yield model and the redundancy selection scheme has been developed in Chapter 3. These are adopted here to assess the yield figures for the FFT superchip. It has already been shown that the pure Poisson model usually leads to a more pessimistic yield figure than is encountered in practice. The models derived by Seed and Stapper are better, fitting observed data reasonably well. By referring to the silicon size of the BPE, Seed's model is adopted here to calculate yield for the BPE. Seed's model has been given in equation (3-5):

$$Y = \frac{1}{1 + DA} \tag{3-5}$$

where D is defect density and A is total critical area on the chip. According to practical experience in custom and semi-custom design, and the actual data from the switch matrix design presented in Section 3.3.2, the total critical area usually lies between 50% and 60% of the total chip area. In the following analysis, the total critical area is assumed to be 50% of the total chip area. Also, the defect density is assumed to be two defects per square centimetre, which is a commonly used figure for modern VLSI technology. With this defect density, the yield of a single BPE is predicted to be 65%, and the probability of successfully making a single route in the matrix is 99.4%. Considering the dependence of all the routes related to one BPE, the probability of successfully routing a single BPE within the switch matrix (16 routes) is predicted to be 91.6% The yield value of the whole FFT superchip with a different number of BPE redundancy is shown in Table 6.2. This shows that adding more than 9 redundant BPEs will not improve the yield dramatically, but instead the cost increases and the figure of merit decreases. However, the maximum yield value achievable by adding redundancy is around 8%, which is two orders of magnitude improvement over the non-redundant case (0.08%). This is paid for by more than a 55% hardware overhead which is clearly large. Although valuable yield improvement can be achieved in comparison to the non-redundant value, the final yield is still low. The question now is: Is there any potential in exploring the architecture which can further improve the yield, but with less hardware overhead? In this regard a hierarchical redundancy scheme may offer a distinct advantages.

6.2.5. Introducing Hierarchical Redundancy

The reason why the yield value can not be dramatically improved, even after a considerable amount of redundancy is introduced into the chip, is that the yield of each BPE is too low. Any single critical defect in a BPE will discard the complete BPE and introduce a large burden on the higher system level's defect tolerance. The system is forced to employ more redundant BPEs. However, after a certain point, the extra,

Table 6.2 Yield Estimation Results†						
Redundancy(BPE)	0	1	2	3	4	5
Yield of matrix	80%	95%	97%	98%	98%	98%
Yield of BPEs	0.1%	0.5%	1.3%	2.5%	3.8%	5%
Chip yield	0.08%	0.47%	1.26%	2.45%	3.7%	4.9%
Figure of Merit	1	5.6	14.1	26.1	37.6	47.6
Redundancy(BPE)	6	7	8	9	10	11
Yield of matrix	98%	98%	98%	98%	98%	98%
Yield of BPEs	6%	6.9%	7.4%	7.7%	7.9%	8.1%
Chip Yield	5.9%	6.8%	7.3%	7.6%	7.7%	7.9%
Figure of Merit	54.9	60.7	62.6	62.7	61.3	60.6

low-yield, redundant BPEs will not give any increase in the overall yield. Improvement can be achieved by lowering the level of the defect-tolerance down into the BPE to increase the yield of each individual BPE. This can be achieved by incorporating a small switch matrix into each BPE. For example, Figure 6.10 shows a schematic of such a two-level redundancy organisation. Inside the BPE, the major yield loss comes from the 12 real multipliers, because they each occupy 80% of the total BPE area. If every real multiplier is considered as an element along the small switch matrix, and all the adders and subtractors are considered as one element along the matrix, then the size of the small matrix is estimated to be about 25 rows and 50 columns, giving a silicon area of around 6.53 mm^2. According to their sizes, each real multiplier will have a yield of 96% and the element of adders and subtractors will also have a yield of 96% (making a

† In this Table, the yield value for the switch matrix does not include the possible yield loss due to critical defects in the control circuits in the matrix. Also, the figure of merit taken is FM_d.

total BPE yield be 65%). If a redundant real multiplier is put into the BPE, this will produce about a 20% hardware overhead for the new BPE including the small switch matrix (11.75 mm^2 extra silicon area versus 54.6 mm^2 for the original silicon size). However, the new BPE yield goes up from 65% to around 90%. A similar table showing the yield values based on this two levels of redundancy is given in Table 6.3. This shows that by employing two levels of redundancy the yield rises dramatically. Even if there is no higher level of redundancy, the chip yield is already 14%, with only a 15% hardware overhead caused by the redundancy in BPE level. Compared to the results in Table 6.2, the hierarchical redundancy scheme needs less overhead, but has higher yield. When the overall yield reaches more than 80% with the higher level of redundancy, the total overhead for both levels is about 55%. This is the same as the overhead required for the highest possible yield in Table 6.2 (single level redundancy case). However, the yield is further improved by an order of magnitude. With reference to the criteria for hierarchical redundancy discussed in Section 3.5, this example lies well inside the scope for which hierarchical redundancy should be applied for better yield enhancement. The results also imply that by introducing an appropriate amount of redundancy at a lower level, more benefit arises than that for the higher level, in terms of overall hardware overhead.

Table 6.3 Yield Estimation Results for Two Levels of Redundancy†

Redundancy(BPE)	0	1	2	3	4	5
Yield of matrix	80%	95%	97%	98%	98%	98%
Yield of BPEs	18%	47%	70%	83%	89%	91%
Chip yield	14.4%	45%	68%	81%	87%	88%
Figure of Merit	1	3.5	4.2	4.8	4.9	4.7

† In this Table, the redundancy in the BPE is fixed, so that the redundancy in the Table refers to the redundancy at system level. Also, the yield value for the switch matrix does not include the possible yield loss due to critical defects in the control circuits in the matrix.

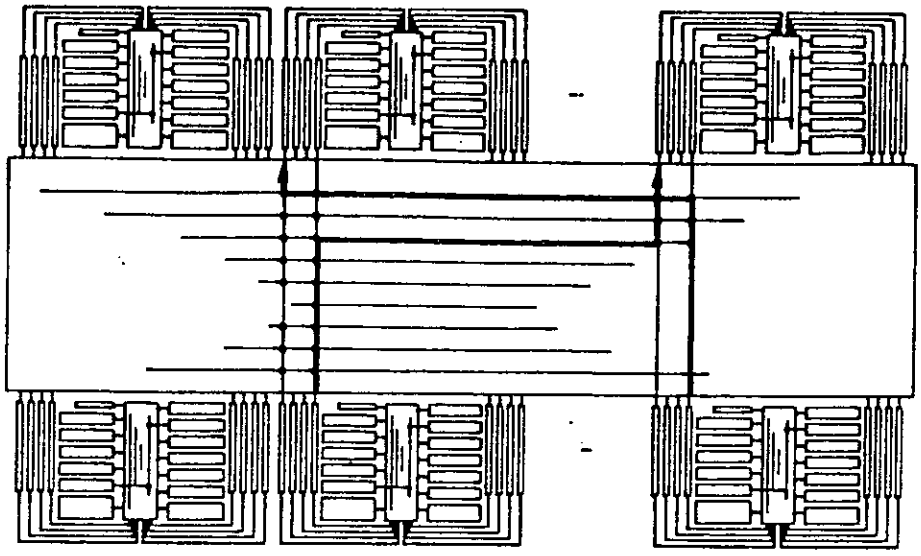

Figure 6.10 A schematic showing two levels of redundancy

6.2.6. Area Reduction

For a 64-point, radix-4, column FFT the size of the switch matrix is 128 by 256 if the inputs are complex signals. In operation, the fact is that both the real and the imaginary part of a sample always go to the same destination. Thus, they can actually share one addressing line in both matrix column and row decoders. They can also share one latching part of the switch cell which controls two transmission gates. By adopting this approach, the area of the switch matrix can be reduced by nearly 50%, as are the matrix row and column decoders. Consequently, the total chip area will be reduced by nearly 10%.

6.3. Expandability of the FFT Superchip

Practically, the transform length for an FFT system is usually more than 64 points. However, it is not practical or economic with current VLSI technology to integrate a full column FFT system with long transform length (say, 256 points) on to one piece of silicon. For FFT chips with rather shorter transform lengths, it is widely desired for them to have expandability in order to easily and arbitrarily form a new FFT system with a longer transform length. The 64-point, full column, FFT presented earlier has such a capability, and can be regarded as a basic building block for constructing larger

column FFT systems.

6.3.1. Forming FFT Systems with Larger Transform Lengths

Let us take an example of a 256-point column FFT system and examine how to build such an FFT system using several 64-point column FFT chips. To perform a 256-point column FFT, there should be 256 traffic lines after every column processing (512 traffic lines for complex input). Four, 64 point column FFT chips should be employed to compose a column with 64, radix-4 butterfly processors. The full column FFT architecture (full column FFT as referred to earlier) requires another twelve, 128 by 256 crossbar switch matrices to implement all the signal traffic. Here, the system complexity may increase dramatically with the increase of the transform length. It may be unmanageable and uneconomic when the transform length becomes reasonably long (say, 1024, where 240 such additional switch matrices are required). In this case, a certain amount of area-time trade-off is necessary. One approach is to reduce the parallelism by introducing parallel-serial column processing, which has slower complexity growth against transform length. Figure 6.11 illustrates such an expanded column FFT machine for 256-point FFT. This consists of four, 64-point column FFT chips and no additional switch matrix is required. Remember that, in the 64-point column FFT, the system can only communicate externally via the switch matrix and a set of bidirectional I/O devices. In this expanded version, the 256 samples are loaded simultaneously into all the 256 FIFOs through an array of switches depicted as "loading switches" in Figure 6.11. These loading switches are very simple. Each slice of these switches takes care of two inputs from outside and two I/O ports, one each from the two neighbouring chips. Figure 6.12 shows the circuit of such a slice. During the loading phase, Φ remains high so that 256 samples can be loaded via the 256 separated switch matrix rows to the local FIFOs. Afterwards, Φ stays low disconnecting the matrix rows from external circuitry and connecting the matrix rows together to form a 64×256 switch matrix (128×512 for complex samples).

A 256-point, radix-4 FFT has four columns. If the physical position of each radix-4 processor is fixed during the transform, and each chip is organised such that the first 16 butterflies are in the first chip, the second 16 are in the second chip, and so on, then inter-chip communication can only be realised through the four linked switch matrices. By studying carefully the Cooley-Tukey 256 point DIF FFT algorithm, shown

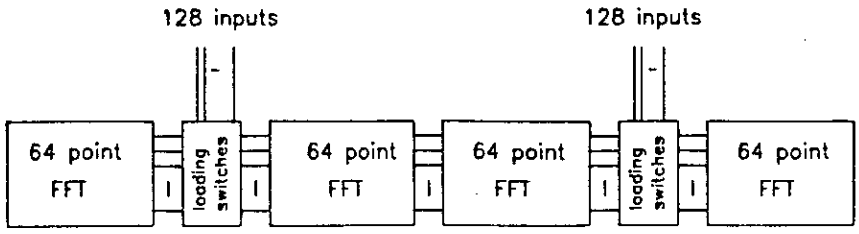

Figure 6.11 Composition diagram of a 256-point column FFT

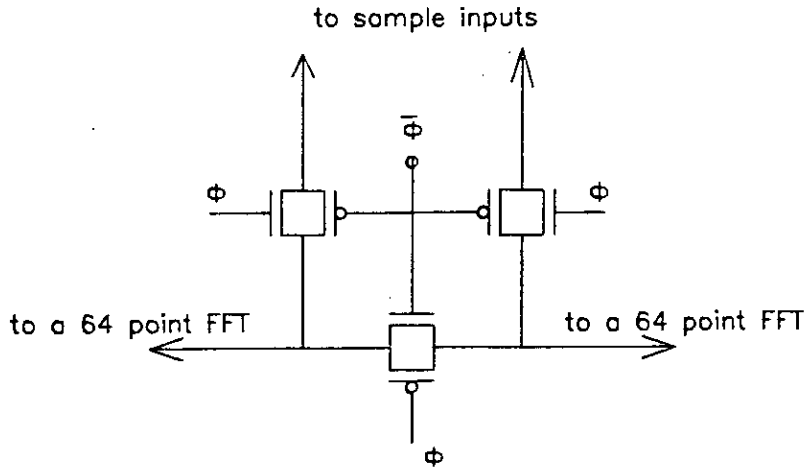

Figure 6.12 One pair of loading switches

diagrammatically in Figure 6.1 for a 64-point transform, it can be found that only the data transfer after the first column computation requires inter-chip (linked switch matrices) communication. Thereafter, the data are only transferred within its own chip (transferred among the 16 BPEs in the chip). By using the tri-state I/O devices on each chip, the chips can be isolated from each other, so that all the 256 samples processed from one column can be routed simultaneously. Looking further to a 1024 point column FFT, this needs 16, 64-point column FFT chips. Only the first column needs data transfer on the full span of the column (16 chips). Thus only 64 points out of 1024 points can be processed at a time for the first column. For the second column, the maximum required span for data transfer is four adjacent chips, so 256 routes can be implemented at a time in the matrix. For the rest of the columns, all the 1024 points can be processed simultaneously. Distinguished by the full column FFT processing, this type of column processing is called *partial column processing*. The Cooley-Tukey algorithm is suitable for partial column processing, because it has localised data transfer while the transform proceeds. Therefore, more parallelism can be achieved for a given size of

switch matrix. If the constant geometry algorithm is used in the partial column processing, then it will be less efficient, because there is no localised data transfer for any column in the transform.

The whole partial column processing can be divided into a serial-parallel processing section and a full parallel processing section. For a 256-point, partial column FFT, all the matrix rows should be connected together to route the traffic for the first column, because the data transfer requires the full span of the column. This is the serial-parallel section in which the computation of the first column is divided into four sequential steps, each of which processes 64 samples in parallel. The results for one group of data are sent to their destinations for the next column processing through the switch matrix. For this situation, it should be assured that data in the FIFOs of those destinations should not be corrupted (overwritten) by incoming data for the next column processing. All the four groups are processed serially in this way. The following full parallel processing is performed for the rest of the three columns by isolating from each other the four crossbar switch matrices in four chips, and routing the traffic locally within each matrix. However, unlike the full column processing where message routing is only performed once for the whole process, the partial column processing requires message routing for all the column computation because the topology of data transfer after each column is different. If we still assume that the system wordlength is equal to the butterfly latency (i.e. the ultimate timing condition is satisfied), then in order to allocate enough time for the message routing, all output data after one column of processing is transferred through the switch matrix and then latched at its destination, instead of being fed into the butterfly processor immediately, as happens in full column processing. After transferring the output data, the next part of the column (or next column) is processed in the butterfly processors and, at the same time, the message routing for this part of the data is carried out in the switch matrix. Figure 6.13 indicates the timing for a 256-point partial column FFT operation, in which the last routing process also brings the final data into natural order. For convenience in later discussion, the maximum number of transferable words of data in one cycle in partial column processing is referred to as the "Base Transform Length", written as N_B. For example, the 256-point partial column FFT discussed above has the base transform length, N_B, of 64. If the time required for processing one sample in the BPE (this is the same as the time required for processing a column in the full column processing) is considered as a time unit, called *a cycle*, then the 256-point partial column FFT requires 14 cycles

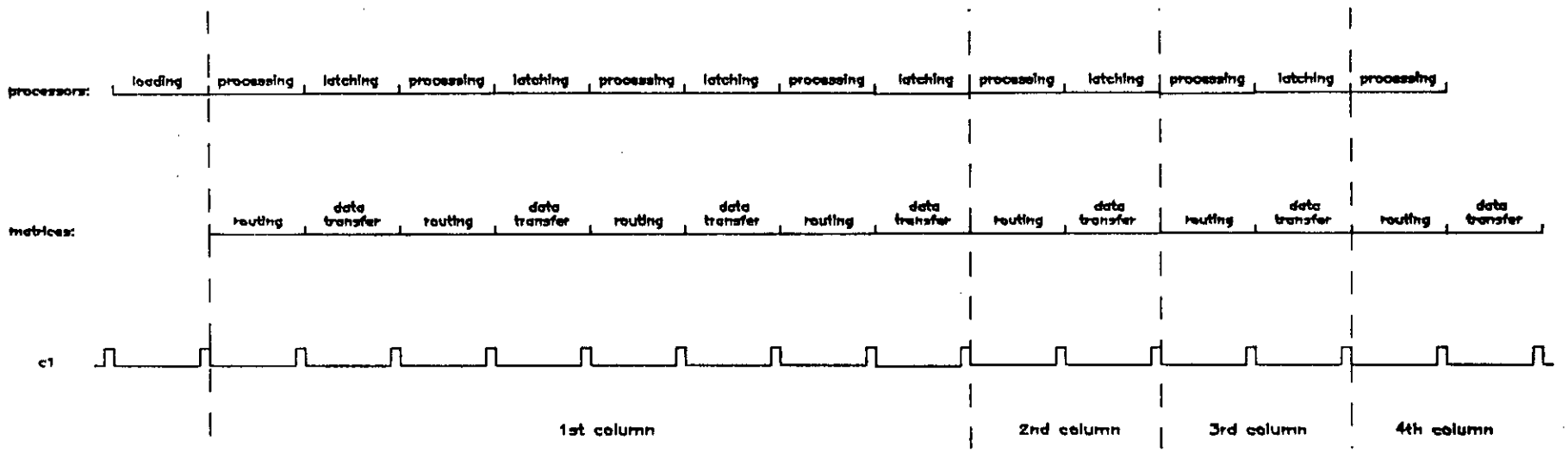

Figure 6.13 Timing overlap of a 256-point partial column FFT

instead of the 4 cycles required by the full column FFT.

6.3.2. Performance Comparison

As discussed above, full column processing is faster than partial column processing, but costs more (for the same transform length). Let us examine the space-time-energy measure to appreciate trade-offs between both organisations. For convenience in following discussion, let the (silicon) area unit be defined as the average silicon area required for processing one sample in the partial column processing (for instance, the silicon area occupied by one BPE and sixteen matrix columns of the matrix, as required for processing four samples). Suppose the system control is distributed amongst the chips such that the ultimate timing condition is satisfied. For full column processing, the time and area required for an FFT with transform length N is $\log_r N$ time units and $\left[N+\Phi N(N-N_B)\right]$ area units, respectively. Φ is the ratio of area unit to area of the extra switch matrices required in the full column processing, which, in this case, is 3.4×10^{-4}, and r is the radix number. For the same situation, partial column processing requires $\left[2log_r N_B + 2 \times \sum_{n=0}^{c} (\frac{1}{4})^n \times (\frac{N}{N_B})\right]$ time units† and N area units, where, c $= \log_r(\frac{N}{N_B}) - 1$. Taking two extremes: Firstly, when N is small, the area and time for full column processing are $O(N)$ and $O(\log_r N)$; the area and time for the partial column processing is $O(N)$ and $O(\log_r N_B + \frac{N}{4N_B})$. Secondly, when N is large and also the condition $N >> N_B$ applies, the area and time for the full column processing are $O(N^2)$ and $O(\log_r N)$; the area and time for the partial column processing are O(N) and $O(N^2)$. Figure 6.14 compares the $area \times time^2$ measure for a full column FFT and two partial column FFTs for different N_B (in which both axes are in log scale). All the data in the diagram have been normalised to the data of a 64-point, full column FFT. The results indicate that the full column processing organisation is generally better than the partial column processing organisation as far as $area \times time^2$ is concerned, particularly when N is large and the condition $N >> N_B$ applies. This result is mainly due to the rapid growth of the time taken by the partial column processing, when transform length

† For the same wordlength and coefficient length, the time unit will be the same for both full column and partial column processing.

becomes large and N_B is kept unchanged. The fact is that during the processing of the first several columns as well as the latching period after processing each column (or part of a column), a certain amount of butterfly processors are idle, while in full column processing, all the butterfly processors are kept busy all the time. To improve the performance for partial column processing when the transform length is large, it is wise to keep $\frac{N}{N_B}$ a small constant (4 is probably the best). For instance, it is better to have 256 rather than 64 as the base transform length in an FFT transform with 1024 or 4096 transform length.

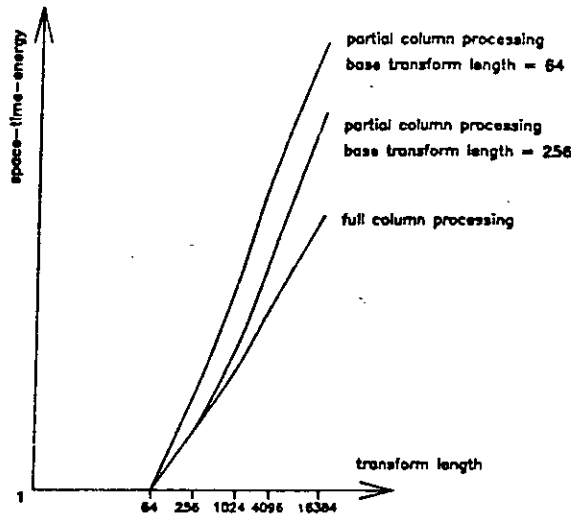

Figure 6.14 Comparison of $area \times time^2$ between the full column and the partial column FFT

The choice of whether to use full column or partial column processing depends upon the hardware availability and the throughput constraint (requirement). Throughput also depends upon the system clock rate, system wordlength and the coefficient length. Suppose that the coefficient for multiplication is 12 bits in length, and the system clock rate is 20 MHz. Then the butterfly latency will be 20 bits. We also assume that the system wordlength is 20 bits as well. Under the ultimate timing condition, one cycle of a column computation is 1 μs. Using these assumptions, Tables 6.4 and 6.5 summarize the computation time and the system complexity taken by both full column and partial column FFT as a comparison, for various transform lengths. Also as a comparison with the performance of several general purpose digital signal processor chips [143], Table 6.6 illustrates their performance for various transform lengths.

Table 6.4 Full Column Processing		
transform length	computation time	transistor count
64 points	3 μsec (3 cycles)	520K
256 points	4 μsec (4 cycles)	1,408K (16 chips)
1024 points	5 μsec (5 cycles)	16,480K (256 chips)
4096 points	6 μsec (6 cycles)	170,368K (4096 chips)

Table 6.5 Partial Column Processing ($N_B = 64$)		
transform length	computation time	transistor count
256 points	14 μsec (14 cycles)	1,000K (4 chips)
1024 points	38 μsec (38 cycles)	8,320K (16 chips)
4096 points	174 μsec (174 cycles)	33,280K (64 chips)

Table 6.6 The FFT Performance of Several DSP Chips			
transform length	7720(NEC)	32020(TI)	56000(Motorola)†
64 points	1.6 msec	0.434 msec	0.147 msec
256 points	–	2.44 msec	0.713 msec
1024 points	77 msec	14.18 msec	5 msec

† The Motorola DSP chip is one of the fastest general purpose single chip DSP with efficient on-chip memory modules.

6.3.3. Message Routing in Partial Column Processing

Extra control circuits are needed to perform message routing for the partial column processing organisation discussed earlier. Note that for each 64-point column FFT chip there is control circuitry for addressing its own switch matrix; this is mainly decoding circuitry. Each set of decoding circuits in a switch matrix can decode addresses from 0 to 255 columns by 8 addressing signals. For a 256-point FFT, there should be 10 addressing signals to address the total number of 1024 columns. Fortunately, the number of columns in each switch matrix is just an integer power of 2. This means, if the 8 least-significant addressing signals out of the 10 addressing signals for the 256-point FFT are taken and applied to a correct switch matrix, then the addressed column should be the correct column which the whole system wants to address. The question arises how to correctly lead these 8 least significant addressing signals to one of the four switch matrices. The answer is that they should be controlled by the remaining two (most significant) addressing signals. Figure 6.15 shows such a control scheme. The two most significant addressing signals are applied to a 2-to-4 line decoder. The four outputs from this decoder control 4 switch arrays. These switch arrays are structurally very simple, and one of them is detailed in Figure 6.16. There are 8 transistors sitting diagonally across the 8 inputs and 8 outputs to form corresponding connections amongst them. They are all controlled by an output signal from the decoder. When this signal is a logic 1, all these 8 switches are closed to lead the 8 least significant addressing signals to this particular switch matrix. Otherwise, the output signal will cause the column decoder in the switch matrix to be in the disabled state, by setting its "NAND" input low. The "set" input in the 2-to-4 line decoder is to set all the outputs high at the same time. This operation is required when the message routing is carried out for the last three columns, during which four traffic routes are implemented in parallel within four switch matrices. The addressing signals to address matrix rows are applied in parallel to all the addressing circuits to make the whole row across the four matrices active.

The generation of the message routing message is the task of the router, treated in Chapter 5. For the constant geometry algorithm, only one message routing is performed. In the partial column processing, the traffic geometry in the switch matrix is changed after each column computation. It has already been assumed that, during the column processing, all the butterfly processors are physically fixed in the chips.

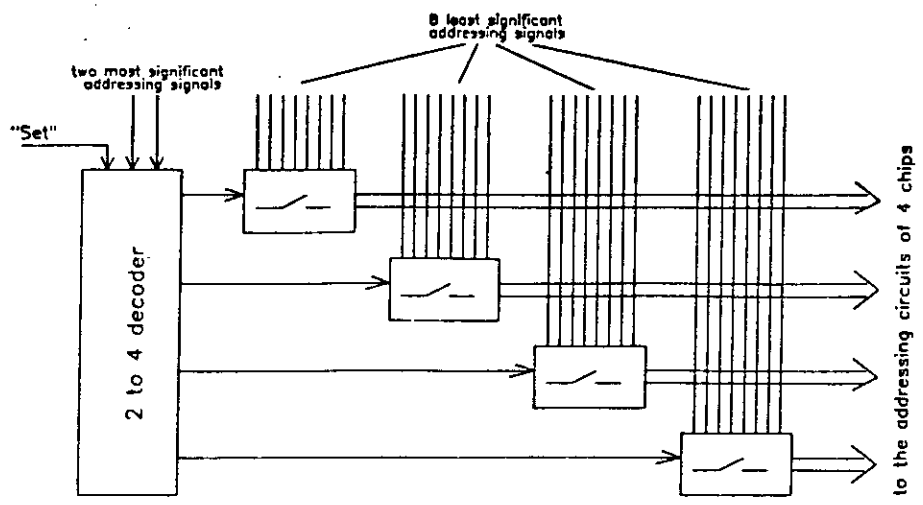

Figure 6.15 Location identification

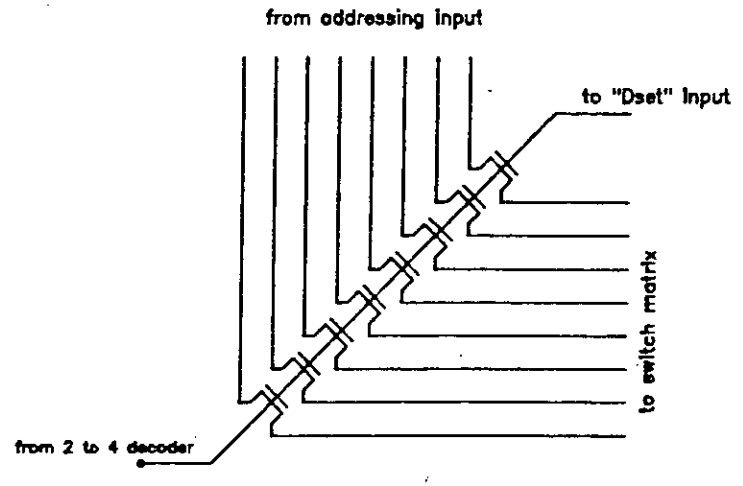

Figure 6.16 A switch array for partial column FFT addressing

Therefore, the route searching procedure in the routing algorithm for the second column onwards should be modified somewhat. Originally, the searching is two dimensional because, for initial routing, the physical position of the processors has to be allocated within the chip (in one dimension), and also the traffic route has to be assigned to one of the matrix rows (in another dimension). Referring to the routing algorithm given earlier, only step 7 (mapping) needs to be changed to be one dimensional searching (fixed column).

6.4. Conclusion

A 64-point, radix-4 column FFT engine implemented with the superchip architecture has been presented. The advantage of having such an FFT engine is that a satisfactory area-time trade-off can be achieved, compared to the full FFT array architecture. Such an advantageous trade-off makes it possible to process an FFT with a longer transform length in parallel. This 64-point column FFT architecture is as flexible as the pipelined FFT architecture, because it can be expanded to a larger FFT transform. However, it is much faster than its pipelined counterpart. Such flexibility mainly results from the use of the crossbar switch matrix. Although integrating the whole column machine on to one piece of silicon makes the total chip area fairly big, employing the switch matrix has also made it possible to introduce redundancy into the chip in such a way as to bring such a design up to an economic level. The transform length of the basic column FFT unit as building block does not have to be 64 points, nor in radix-4. These should be adjusted according the technology and the application.

The column FFT system is only one example of superchip architecture. It was chosen here, to evaluate the potential for defect-tolerance and system reconfigurability. There are, of course, many other potential applications such as matrix manipulation, image processing, and general purpose multiprocessing, etc, where a similar advantage may be achievable.

CHAPTER 7

CONCLUSIONS AND RECOMMENDATIONS FOR FURTHER RESEARCH

7.1. Summary

For future signal processing and scientific computational tasks, the superchip architecture proposed in this thesis offers higher throughput, flexibility and lower cost than conventional approaches. This is achieved by integrating a large crossbar switch matrix which connects a large number of PEs onto a single piece of silicon. In such an environment, the crossbar switch matrix as a communication network plays a key role. The thesis has assessed the potential of such a structure in large integrated systems and covers the defect/fault tolerant capability and the system reconfigurability, both at the top level of the hardware design methodology and at a lower level of software support.

Two types of crossbar switching network were examined: a *normal switch matrix* and a *duplex switch matrix*. The reason for choosing the crossbar network is provided by the analysis and comparison of the crossbar switching network approach with other interconnection networks and techniques available for large, high performance ICs. In the superchip, large numbers of PEs are fully connected to each other through a crossbar switch matrix. By carefully allocating processing tasks among these PEs the system throughput can be maximised, resulting in efficient parallel processing. The example of a 64-point column FFT machine, discussed in Chapter 6, indicated that such system is capable of approximately 150 million arithmetic operations per second, and only occupies an area of 35×35 mm^2. More powerful systems can be achieved by extending the approach to an entire silicon wafer using the same processing technology, or increasing the density using smaller geometry processing technology.

Initial results for the prototype switch matrix design vindicate the functionality of the switch matrix designed in this research. They also demonstrate the good

performance characteristics of such a design. For example, the signal delay through the switch matrix is expected to be around 30 to 50 nsec with little distortion (pulse width variation). The superchip can easily be programmed through a set of addressing circuits and, thus, the user will have more flexibility to change the configuration during operation. Defect/fault tolerance is achieved by avoiding faulty switches or faulty PEs during traffic routing. Such a tolerant capability becomes very efficient in the duplex matrix architecture where there exist multiple paths between any two communicating ports. The column FFT machine discussed as a case study is a good example of realising dynamic reconfiguration and defect tolerance in the crossbar switch matrix.

Yield models and a routing algorithm have also been developed for the superchip and have been presented in this thesis. Yield modelling is crucial for the superchip approach because any system implemented using it is potentially a large area silicon chip. The cost of a system reflects significantly the superchip yield figure. More importantly, the yield models lay the foundation for estimating optimal redundancy for a system. In a large silicon chip, like the superchip, processing defects are inevitable. Defect tolerance through hardware redundancy is necessary to improve yield and therefore reduce the cost of implementation. However, from a cost-effectiveness point of view, the hardware overhead should be kept to a minimum, with respect to a required yield. In other words, every redundant element introduced should be cost effective, or worth paying for by such an area overhead, in return for the amount of yield improvement. Such cost effectiveness is estimated by the introduction of a "figure of merit", which indicates the optimal amount of redundancy for a particular system.

The amount of redundancy is one factor which contributes to the cost-effectiveness of the superchip implementation of a system. There are two other important factors which also relate to yield and cost-effectiveness. One is system partitioning. For a particular system, there will be several different ways to map the system into the superchip architecture, depending on how the system partitions into the PEs of the superchip. Different partitioning results in differently sized PEs and so a different overall size of the superchip, resulting in different yield values. For a particular system, equation (3-41) quantifies the benefit of using the superchip architecture for various system partitions against its non-programmable implementation (without using the switch matrix). The peak value indicates the most beneficial partition of the system. However, this measure together with the figure of merit is not in itself able to justify cost-effectiveness.

This is because the benefits are obtained in comparison with either a non-redundant or non-programmable system. The real potential benefit due to different redundancy has not been fully explored and compared. This leads to the other important factor: the distribution of redundancy amongst different system levels. Primarily, the degree of system partitioning can ensure that the partition (very likely from the natural hierarchy of the design) falls inside the beneficial region (given in Figures 3.20 and 3.22). This also determines the size of the matrix. Then the careful distribution of the amount of redundancy for any particular size can be made to maximise the yield value, for which the figure of merit for the amount of redundancy can be taken as an indicator of the cost-effectiveness for each level.

The message routing algorithm introduced in this thesis was designed to construct a desired system from a partially defective superchip. It classifies all the faulty elements and their fault conditions. Good PEs are then connected through the switch matrix in such a way that every route being searched gathers all the related local and global advantages and disadvantages as to where to embed the route in orer to guarantee the maximum possibility of success. This has been achieved through the definition of a goodness function. As proved in Appendix B, the message routing problem in a partially good switch matrix is NP-complete. To find a solution for this NP-complete problem, the goodness function is not only used to define goodness values for each route, but is also used to define a practical searching heuristic for the problem. the goodness function defines a region where the greatest possibility of finding a solution seems likely. Various tests on different fault densities and fault locations were carried out for this approach. The results were satisfactory.

7.2. Recommended Further Research

Further research directions either related to or stemming from this thesis are discussed in the following five sub-sections.

7.2.1. Increasing Testability of the Switch Matrix Design

It has been shown that the current switch matrix prototype design can not get 100% coverage for functional test. The problem lies in the decoding circuits where 100% coverage of functional test can not be achieved. This is because the outputs of these circuits can not be read directly. Incorporation of an observation pad at each

output seems too expensive, leading to an unaccepted pin count. The best way to solve the problem is to introduce a shift register chain between the column decoder and the switch matrix, as well as between the row decoder and the switch matrix. Figures 7.1 and 7.2 show such an implementation for the column decoder and the row decoder, respectively. Two complementary outputs from the column decoding circuits (including NAND and Tesc circuit) are switched by two multiplexers. During testing, these outputs are fed into the shift registers chain. They are then shifted out to be checked. During system operation, these outputs are fed back to the switch matrix. Similarly, for the row decoder case, outputs from the decoder are switched to a shift register through a multiplexer during testing, and fed back to the switch matrix during operation. By introducing shift registers into the switch matrix, control circuit testing becomes possible for a low pin-count overhead. Furthermore, they can separate switch cell testing from the decoding circuits so that there will be no confusion about fault location. Also, address patterns can be shifted into the shift registers and applied to the switch cells directly circumventing use of the decoding circuits. This can offer full observability and controllability of the switch cells.

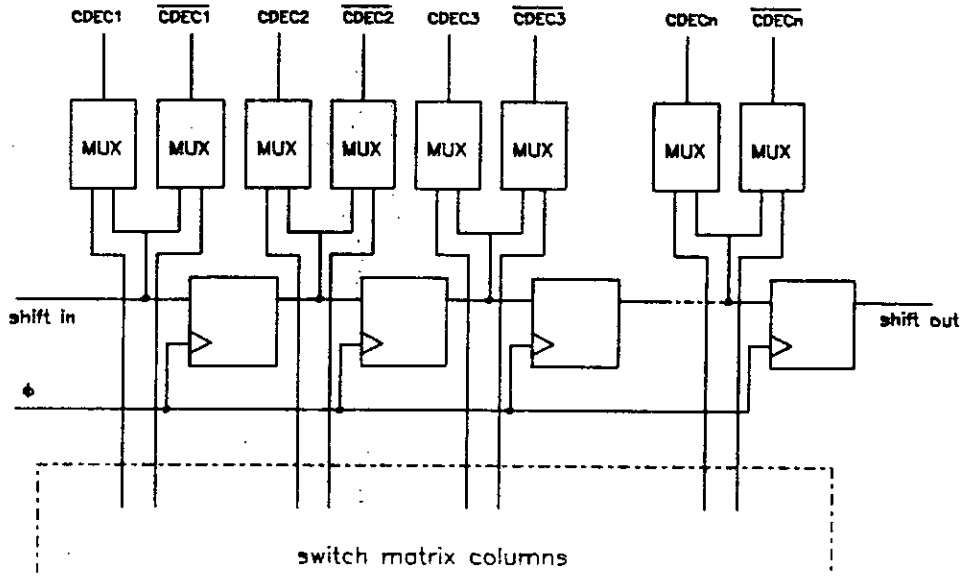

Figure 7.1 A schematic showing separation of the column decoder and the switch matrix for test observability and controllability.

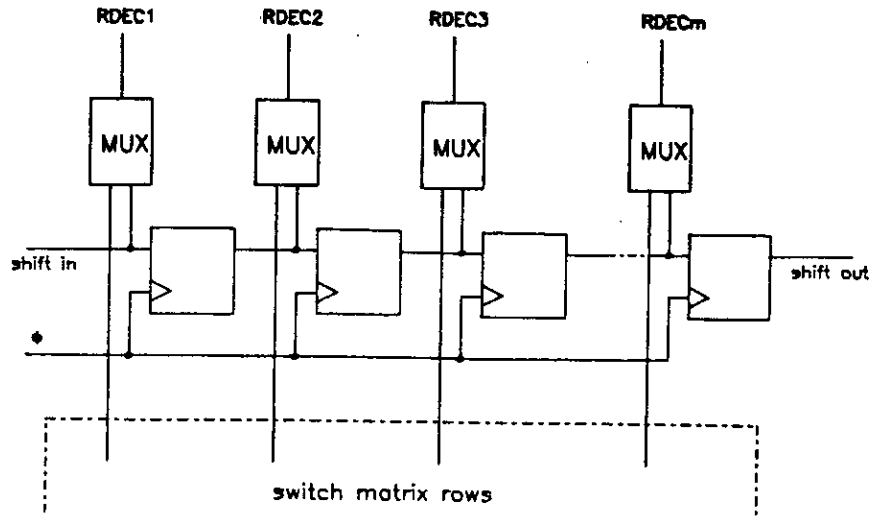

Figure 7.2 A schematic showing separation of the row decoder and the switch matrix for test observability and controllability.

7.2.2. More Efficient Use of the Switch Matrix

Two types of switch matrix, the normal matrix and the duplex matrix, have a common feature. There is only one net allowed in each matrix row. Let the distance between the left-most point and the right-most point of a net in a matrix row be called the *span* of the net. If the span of a net implemented on a matrix row is very small, the usage of the matrix row will be very low, Thus a large proportion of the matrix row does not effectively carry signals. For a more efficient use of the switch matrix, it is desirable for each matrix row to carry more than one net, particularly when the span of these nets is very small. This can result in a smaller switch matrix for the same system. For example, Figure 7.3 shows a modified structure in which several switches are introduced into each of the matrix rows. By turning off some of these switches, more than one net can be embedded in a single matrix row. Each of these switches can be addressed using an output line of the column decoder (shown by the dotted lines in Figure 7.3). The question is how many such switches are needed for a particular switch matrix, what kind of relationship they should have and how to distribute these switches inside the switch matrix.

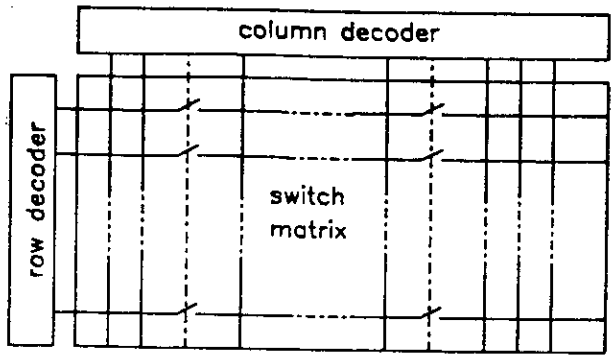

Figure 7.3 A modified switch matrix structure

7.2.3. Hierarchical Redundancy for the Switch Matrix

The results from Chapters 3 and 5 showed that hierarchical redundancy may be much more effective than the simple one-level redundancy. More research is needed on this subject. For example, for a particular system to be implemented in the superchip architecture, what amount of hierarchical redundancy is required and how should it be distributed amongst various levels of the system in terms of system complexity and partitioned PE complexity?

7.2.4. Self-Testing

It has already been mentioned, at the beginning of the thesis, that a larger system will suffer more transient (soft) and permanent failures during field operation. This gives rise to a very serious question: how can a large system efficiently overcome this potential reliability problem? Efficient self-testing during field operations seems to be the right way to tackle the problem. It is desirable that both the PEs and the switch cells in the superchip have their own self-test circuits. Self-testing is performed periodically during the field operation, so that the transient operational faults can be discovered and any related faulty element can then be bypassed for the next computational cycle.

7.2.5. Superchip Array

An expanded column FFT system organised by connecting several superchips together through the crossbar switch matrix has already been given as an example for the superchip array. It uses the expandability of the interconnection network to form larger arrays. It is reasonable to organise a system in this way, i.e. a full column FFT

machine when fast, frequent, global communication among all the PEs is required. However, such an organisation is generally expensive because the interconnection cost grows faster than the PE cost. For some applications where instant-response communication is not of primary importance, or where there are several local communication 'hot-spots', in these cases such global, instant-response communication may not always be suitable or cost-effective. It will be appropriate for some applications to have other organisations with lower system complexity which achieves the required system performance. Various organisations for connecting an array of superchips could be examined. For example, it might be useful to introduce several direct inter-PE links inside a superchip, or among the superchips, to solve local hot-spot contentions of inter-PE communication.

REFERENCES

1. J. P. Beraud, "Signal Processor Chip Implementation," *IBM J. Res. Develop.* Vol. 29 pp. 140 - 146 (March 1985).

2. T. Nishitani, I. Kuroda, and Y. Kawakami et al, "Advanced Single-Chip Signal Processor," *Proceedings on Int. Conf. on Acoustics, Speech, and Signal Processing*, pp. 409 - 412 Tokyo, Japan, (1986).

3. J. W. Beyes et al, "A 32-bit VLSI CPU Chip," *IEEE J. Solid-State Circuits* SC-16 pp. 537 - 541 (October 1981).

4. Inmos Corp., *IMS T424 Transputer*, Advanced Info., Colorado Springs, CO (1984).

5. Y. Ueoka, C. Minagawa, M. Oka, and A. Ishimoto, "A defect-Tolerant Design for Full-Wafer Memory LSI," *IEEE J. of Solid-State Circuits* SC-19(3) pp. 319 - 324 (June 1984).

6. J. M. Wilkinson and N. MacDonald, "A commercial Approach to Wafer Scale Integration," *The Proc. of International Workshop on Wafer Scale Integration*, Southampton, England, (July 1985).

7. J. W. Lathrop, R. Scot Clark, J. E. Hull, and R. M. Jennings, "A Discretionary Wiring System as the Interface Between Design Automation and Semiconductor Array Manufacture," *Proceedings of the IEEE* 55(11) pp. 1988 - 1997 (Nov. 1967).

8. J. C. Hunter, "Database Retrieval Using Superchip," *Int. Sym. on Advanced Memory Products*, (1976).

9. C. A. Finnila and H. H. Love, "The Associative Linear Array Processor," *IEEE Trans. on Computers* 26 pp. 112 - 115 (1977).

10. D. L. Peltzer, "Wafer Scale Integration: the limits of VLSI?," *VLSI Design*, pp. 43 - 47 (Sept. 1983).

11. Michael J. Newman, "Is WSI Too Big? Selecting an Appropriate Component Size for Redundant Logic," *VLSI 85, International Conference on VLSI*, pp. 97 - 106 Tokyo, Japan, (August 26-28, 1985).

12. C. A. Neugebauer, "Approaching Wafer Scale Integration from the Packaging Point of View," *Proceedings of Int. Conf. on VLSI in Computers*, pp. 115 - 120 (1984).

13. C. A. Neugebauer, "Comparison of VLSI Packaging Approaches to Wafer Scale Integration," *Proceedings of IEEE 1985 Custom Integrated Circuits Conference*, pp. 32 - 37 (1985).

14. Y. Tamir and C. H. Sequin, "Error Recovery in Multicomputers Using Global Checkpoints," *Proc. 1984 Int. Conf. on Parallel Processing*, pp. 32 - 41 (August 1984).

15. J. Anderson and F. Macri, "Multiple Redundancy Applications in a Computer," *Proc. 1967 Annual Sym. on Reliability*, pp. 553 - 562 (Jan. 1967).

16. C. S. Draper Lab., "Advanced Information Processing System (AIPS) System Specification," *prepared for Johnson Space Center by C. S. Draper Laboratory, Cambridge, MA*, (May 15 1984).

17. Gabriele Saucier, "Terminology for WSI Design Test," *Proceedings of I.F.I.P. Workshop on Wafer Scale Integration*, Grenoble, France, (17-19 March, 1986).

18. Sudhakar Yalamanchili and J. K. Aggarwal, "Reconfiguration Strategies for Parallel Architectures," *Computer*, pp. 44 - 61 (Dec. 1985).

19. Tulin E. Mangir, "Use of On-Chip Redundancy for Fault-Tolerant Very Large Scale Integrated Circuit Design," Ph.D. Thesis, University of California, Los Angeles (1981).

20. C. H. Stapper, A. N. McLaren, and M. Dreckmann, "Yield Model for Productivity Optimization of VLSI Memory Chips with Redundancy and Partially Good Product," *IBM J. Res. Develop.* 24(3) pp. 398 - 409 (May 1980).

21. C. H. Stapper, "Modeling of Integrated Circuit Defect Sensitivities," *IBM J. Res. Develop.* 27 pp. 549 - 557 (Nov. 1983).

22. C. H. Stapper, "Modeling of Defects in Integrated Circuit Photolithographic Patterns," *IBM J. Res. Develop.* 28(4) pp. 461 - 475 (July 1984).

23. R. Micheal Tanner, "Fault-Tolerant 256K Memory Designs," *IEEE Trans. on Computers* C-33(4) pp. 314 - 322 (April 1984).

24. Yoshinori Okejima and Kazuhiro Toyoda, et al., "64K ECL RAM with Redundancy," *IEEE Int. Solid-State Circuits Conference, Dig. of Paers,* pp. 48 - 49 (Feb. 1985).

25. Masaki Kumanoya and Kazuyasu Fujishima et al., "A 90ns 1Mb DRAM with Multi-Bit Test Mode," *Proceedings fo IEEE Int. Solid-State Circuits Conference,* pp. 240 - 241 (Feb. 1985).

26. Howard L. Kalter and P. Coppens et al., "An Experimental 80ns 1Mb DRAM with Fast Page Operation," *IEEE Int. Solid-State Circuits Conference, Dig. of Papers,* pp. 248 - 249 (Feb. 1985).

27. B. J. Donlan and G. F. Taylor et al., "Wafer Scale Integration Using Discretionary Microtransmission Line Interconnections," *Proceedings of Int. Workshop on Wafer Scale Integration,* Southampton, England, (July 1985).

28. J. I. Raffle et al., "A Demonstration of Very Large Area Integration Using Laser Restructuring," *Proceedings of IEEE Int. Symp. on Circuits and Systems,* pp. 781 - 783 (1983).

29. J. I. Raffle et al., "A Wafer Scale Digital Integrator Using Restructurable VLSI," *IEEE Journal of Solid-State Circuits* SC-20(1) pp. 399 - 406 (Feb. 1985).

30. J. I. Raffle, "The RVLSI Approach to Wafer Scale Integration," *Proceedings of Int. Workshop on Wafer Scale Integration*, Southampton, England, (July 1985).

31. S. L. Garverick and E. A. Pierce, "A Single Wafer 16-Point 16 MHz FFT Processor," *Proceedings of IEEE CICC*, pp. 244 - 247 (1983).

32. F. M. Rhodes, "Application of RVLSI to Signal Processing," *Proceedings of Int. Workshop on Wafer Scale Integration*, Shouthampton, England, (July 1985).

33. D. J. Ehrlich et al., "Techniques for Improving Engineering Productivity of VLSI Designs," *IBM J. of Res. and Develop.* 25 pp. 107 - 115 (May 1981).

34. R. P. Cenker, D. G. Clemons, and W. R. Petrizzi et al., "A Fault Tolerant 64K Dynamic Random Access Memory," *IEEE Trans. on Electron Devices* ED-26(6) pp. 853 - 860 (June 1979).

35. R. T. Smith, "Using a Laser Beam to Substitute Good Cells for Bad," *Electronics*, pp. 131 - 134 (28 July, 1984).

36. R. T. Smith, J. D. Chlipala, and J. F. M. Bindels, "Laser Programmable Redundancy and Yield Improvement in a 64K DRAM," *IEEE Journal of Solid-State Circuits* SC-16(5) pp. 506 - 513 (Oct. 1981).

37. O. Minato et al., "A Hi-CMOS 8K×8 bit static RAM," *IEEE Journal of Solid-State Circuits* SC-17(5) pp. 793 - 797 (Oct. 1982).

38. K. Ochii and K. Hashimoto et al., "An Ultralow Power 8K×8 bit full CMOS RAM," *IEEE Journal of Solid-State Circuits* SC-17(5) pp. 798 - 803 (Oct. 1982).

39. Ron Taylor and Mark Johnson, "A 1Mb CMOS DRAM with a Divided Bitline Matrix Architecture," *IEEE Int. Solid-State Circuits Conference, Dig. of Papers*, pp. 242 - 243 (Feb. 1985).

40. D. C. Shaver, *J. Vac. Sci.Technol.*, p. 1084 (1983).

41. D. C. Shaver et al., "Electron-Beam Programmable 128K bit Wafer Scale EPROM," *IEEE Electron Device Lett.* EDL-4 pp. 153 - 155 (May 1983).

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

43. Jeff Fried, "Wafer-Scale Integration of Pipelined Processors," *Proceedings of Int. Conf. on VLSI in Computers*, pp. 611 - 615 (1984).

44. P. Girard, F. M. Roche, and B. Pistoulet, "Electron Beam Effects on VLSI MOS Conditions for Testing and Reconfiguration," *Proceedings of I.F.I.P. Workshop on Wafer Scale Integration*, Grenoble, France, (March 1986).

45. Russell C. Aubusson and Ivor Catt, "Wafer-Scale Integration--A Fault-Tolerant Procedure," *IEEE J. of Solid-State Circuits* SC-13(3) pp. 339 - 344 (June 1978).

46. M. J. Shute, "The Role of Simulation on the Study of Multiprocessor, Control Flow and Data Flow Systems," Ph.D. thesis, University of London (1983).

47. M. J. Shute and P. E. Osmon, "COBWEB - A Reduction Architecture," *Proceedings of Int. Workshop on Wafer Scale Integration*, Southampton, England, (July 1985).

48. R. A. Evans, J. V. McCanny, and K. W. Wood, "Wafer Scale Integration Based on Self-Organisation," *Proceedings of Int. Workshop on Wafer Scale Integration*, Southampton, England, (July 1985).

49. A. Rushton and C. R. Jesshope, "The Reconfigurable Processor Array - An Architecture in Need of WSI," *Proceedings of Int. Workshop on Wafer Scale Integration*, Southampton, England, (July 1985).

50. G. Chevalier and G. Saucier, "A Programmable Switch for Fault Tolerant Wafer Scale Integration of Processor Arrays," *Proceedings of Int. Workshop on Wafer Scale Integration*, Southampton, England, (July 1985).

51. R. Schuck and M. Glesner, "A WSI-System for the Computation of the Two-Dimensional Fast Fourier Transform," *Proceedings of Int. Workshop on Wafer Scale Integration*, Southampton, England, (July 1985).

52. Manolis G. H. Katevenis and Miriam G. Blatt, "Switch Design for Soft-Configurable WSI Systems," pp. 197 - 219 in *Proc. of 1985 Chapel Hill Conference on VLSI*, ed. Henry Fuchs, Computer Science Press (1985).

53. T. Kawada et al., "A Pattern Matching Processor Array with Defect Tolerance," *1986 Solid-State Circuits Conf. Digest of Tech. Papers*, pp. 90 - 91 (Feb. 1986).

54. Yuen H. Chan et al, "A 3ns 32K Bipolar RAM," *1986 Solid-State Circuits Conf. Digest of Tech. Papers*, pp. 210 - 211 (Feb. 1986).

55. Herbert Stopper, "A Wafer with Electrically Programmable Interconnections," *Int. Solid-State Circuits Conf., Digest of Tech. Papers*, pp. 268 - 269 (February 1985).

56. Electronics Week, April 8, 1985.

57. R. R. Johnson, "The Significance of Wafer Scale Integration in Computer Design," *Proc. of IEEE Int. Conf. on VLSI in Computers*, pp. 101 - 105 (1984).

58. R. R. Johnson, a talk at I.F.I.P Workshop on Wafer Scale Integration, , Grenoble, France (March 1986).

59. G. F. Tayor and B. J. Donlan et al., "?," *Proceedings of IEEE 1985 Custom Integrated Circuits Conference*, pp. 55 - 58 (1985).

60. R. W. Johnson et al, "Hybrid Silicon Wafer Scale Packaging Technology," *1986 Solid-State Circuits Conf. Digest of Tech. Papers*, pp. 166 - 167 (Feb. 1986).

61. D. F. Calhoum and G. Wolfe, "Status and Future directions of Pad Relocation Full Wafer LSI," *Proc. of Electronic Components Conf.*, pp. 7 - 14 (1973).

62. J. I. Raffel, "On the Use of Nonvolatile Programmable Links for Restructurable VLSI," *Proc. Caltech Conf. VLSI*, pp. 95 - 104 (1979).

63. Yukun Hsia et al, "Adaptive wafer-Scale Integration," *Proc. of 11th Conf. Solid-State Devices*, pp. 193 - 202 (1979).

64. C. Jesshope and L. Bentley, "A Low-Cost Restructuring Techniques for WSI," *IEE Colloquium on "Fault Tolerant ICs/Wafer Scale Integration"*, (17 Feb, 1986).

65. K. J. Thurber, "Interconnection Networks - A Survey and Assessment," *AFIPS Conference Proceedings* 43 pp. 909 - 919 (1974).

66. K. J. Thurber, "Circuit Switching Technology: A State-of-the-Art Survey," *Proc. of Compcon Fall 1978*, pp. 116 -124 (Sept. 1978).

67. H. J. Siegel, R. J. McMillen, and P. T. Mueller, "A Survey of Interconnection Methods for Reconfigurable Parallel Processing Systems," *AFIPS Conference Proceedings* 48 pp. 387 - 400 (1979).

68. G. M. Masson, G. C. Gingher, and S. Nakamura, "A Sampler of Circuit Switching Networks," *Computer* 12(6) pp. 32 - 48 (June 1979).

69. H. J. Siegel, "Interconnection Networks for SIMD Machines," *Computer* 12(6) pp. 57 - 66 (June 1979).

70. C. Wu and T. Feng, "A VLSI Interconnection Network for Multiprocessor Systems," *Digest of Compcon Spring*, pp. 294 - 298 (1981).

71. Tse-yun Feng, "A Survey of Interconnection Networks," *Computer*, pp. 12 - 27 (Dec. 1981).

72. C. Wu and T. Feng, "On a Class of Multistage Interconnection Networks," *IEEE Trans. on Computers* C-29(8) pp. 694 - 702 (August 1980).

73. W. A. Wulf and G. C. Bell, "Cmm - A Multi-mini-processor," *Proc. AFIPS 1972 Fall Jt. Computer Conf.* 41 pp. 765 - 777 AFIPS Press, (1972).

74. W. A. Wulf et al., "HYDRA: the Kernel of a Multiprocessor Operating System," *Communication ACM* 17(6) pp. 337 - 345 (June 1974).

75. W. A. Wulf and R. Levin, "A Local Network," *Datamation*, pp. 47 - 50 (Feb. 1975).

76. T. Feng, "Parallel Processing Characteristics and Implementation of Data Manipulating Functions," Rome Air Development Center Report, RADC-TR-73-189 (July 1973).

77. C. Wu and T. Feng, "On a Distributed-Processor Communication Architecture," *Proc. Compcon Fall 1980*, pp. 599 - 605 (1980).

78. C. Wu, T. Feng, and M. Lin, "Star: A Local Network System for Real-Time Management of Imagery Data," *IEEE Trans. on Computers* C-31 pp. 923 - 933 (Oct. 1982).

79. L. R. Goke and G. J. Lipovski, "Banyan Networks for Partitioning Multiprocessing Systems," *Proc. First Annual Computer Architecture Conf.*, pp. 21 -28 (Dec. 1973).

80. D. H. Lawrie, "Access and Alignment of Data in an Array Processor," *IEEE Trans. on Computers* C-24(12) pp. 1145 - 1155 (Dec. 1975).

81. J. H. Patel, "Processor-Memory Interconnections for Multiprocessors," *Proc. 6th Annual symp. Computer Architecture*, pp. 168 - 177 (April 1979).

82. C. Shannon, "Memory Requirements in a Telephone Exchange," *Bell Systems Tech. Journal* 29 pp. 343 - 349 (1950).

83. G. B. Adams III and H. J. Siegel, "The Extra Stage Cube: A Fault-Tolerant Interconnection Network for Supersystems," *IEEE Trans. on Computers* C-31(5) pp. 443 - 454 (May 1982).

84. G. B. Adams III and H. J. Siegel, "Modifications to Improve the Fault Tolerance of the Extra Stage Cube Interconnection Network," *Proc. of Int. Conf. on Parallel Processing*, pp. 169 - 173 (August 1984).

85. C. S. Raghavendra and Anujan Varma, "Fault-Tolerant Multiprocessors with Redundant-Path Interconnection Networks," *IEEE Trans. on Computers* C-35(4) pp. 307 - 316 (April 1986).

86. Christos J. Georgiou, "Fault-Tolerant Crosspoint Switching Networks," *Proc. of 14th Int. Sym. on Fault-Tolerant Computing*, pp. 240 - 245 (1984).

87. M. A. Franklin, "VLSI Performance Comparison of Banyan and Crossbar Communication Networks," *IEEE Trans. on Computers* C-30 pp. 283 - 291 (Appril 1981).

88. Peter B. Denyer, David Renshaw, *VLSI Signal Processing: a bit-serial approach*, Addison-Wesley Publishers Ltd (1985).

89. A. F. Murray, P. B. Denyer, and D. Renshaw, "Self-Testing in Bit-Serial VLSI Parts: High Coverage at Low Cost," *Proc. Int. Test Conf.*, pp. 260 - 268 (1983).

90. R. F. Lyon, "A Bit-Serial VLSI Architectural Methodology for Signal Processing," pp. 131 - 140 in *VLSI 81*, ed. J. P. Gray, Academic Press (1981).

91. B. T. Murphy, "Cost-Size Optima of Monolithic Integrated Circuits," *Proc. of IEEE* 52 pp. 1537 - 1545 (Dec. 1964).

92. R. B. Seeds, "Yield, Economic, and Logistic Model for Complex Digital Arrays," pp. 60 in *1967 IEEE Int. Conv. Rec.*, (Oct. 1967).

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

94. C. H. Stapper, "Defect Density Distribution for LSI Yield Calculations," *IEEE Trans. on Electron Devices*, pp. 655 - 657 (July 1973).

95. C. H. Stapper, "On a Composite Yield Model to the IC Yield Problem," *IEEE J. of Solid-State Circuits* SC-10 pp. 537 - 539 (1975).

96. C. H. Stapper, "Yield Model for Fault Clusters within Integrated Circuits," *IBM J. Res. Develop.* 28(5) pp. 636 - 640 (Sept. 1984).

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

98. S. M. Hu, "On Yield Projection for VLSI and Beyond; I - Analysis of Yield Formulas," *Electron Devices, Issue 69*, pp. 4 - 10 (Mar. 1984).

99. C. H. Stapper, "Comments on "Some Considerations in the Formulation of IC Yield Statistics", *Solid-State Electronics* 24 pp. 127 - 131 (1981).

100. C. H. Stapper, "Yield Statistics for Large Area ICs," *1986 Digest of Technical Papers of IEEE International Solid-State Circuits Conference*, pp. 168 - 169 (Feb. 1986).

101. C. H. Stapper, "The Defect-Sensitivity Effect of Memory Chips," *IEEE Journal of Solid-State Circuits* SC-21(1) pp. 193 - 198 (Feb. 1986).

102. O. Paz and T. R. Lawson, "Modification of Poisson Statistics: Modeling Defects Induced by Diffusion," *IEEE J. of Solid-State Circuits* SC-12(5) pp. 540 - 546 (Oct. 1977).

103. G. E. Moore, "What Level of LSI Is Best for You?," *Electronics* 43 pp. 126 - 130 (Feb. 1970).

104. A. Gupta, W. A. Porter, and J. W. Lathrop, "Defect Analysis and Yield Degradation of Integrated Circuits," *IEEE J. of Solid-State Circuits* SC-9(3) pp. 96 - 103 (June 1974).

105. A. G. F. Dingwall, "High Yield-Processed Bipolar LSI Arrays," pp. 82 in *1968 Int. Electron Device Meeting Tech. Dig.*, (Oct. 1968).

106. R. M. Warner, "Applying a Composite Model to the IC Yield Problem," *IEEE J. of Solid-State Circuits* SC-9(3) pp. 86 - 95 (June 1974).

107. R. M. Warner, "A Note on IC-Yield Statistics," *Solid-State Electronics* 24(11) pp. 1045 - 1047 (1981).

108. C. H. Stapper, "The Effects of Wafer to Wafer Defect Density Variations on Integrated Circuit Defect and Fault Distributions," *IBM J. Res. Develop.* 29 pp. 87 - 97 (Jan. 1985).

109. C. H. Stapper, "LSI Yield Modeling and Process Monitoring," *IBM J. Res. Develop.* 20 pp. 228 - 234 (May 1976).

110. C. H. Stapper, "Yield Model for 256K RAMs and Beyond," *1982 Digest of Technical Papers of IEEE International Solid-State Circuits Conference*, pp. 12 - 13 (Feb. 1982).

111. P. Banerjee and J. A. Abraham, "Fault Characterization of VLSI MOS Circuits," *Proc. IEEE International Conference on Circuits and Computers*, pp. 564 - 568 (1982).

112. C Timoc, M. Buehler, T. Griswold, and C. Pina, et al, "Logical Models of Physical Failures," *Proceedings of 1983 International Test Conference*, pp. 546 - 553 (1983).

113. J. Galiay, Y. Crouzet, and M. Vergniault, "Physical Versus Logical Fault Models MOS LSI Circuits: Impact on Their Testability," *IEEE Trans. on Computers* C-29(6) pp. 527 - 531 (June 1980).

114. R. L. Wadsack, "Fault Modeling and Logic Simulation of CMOS and MOS Integrated Circuits," *The Bell System Technical Journal* 75(5) pp. 1449 - 1474 (May-June 1978).

115. A. V. Ferris-Prabhu, "Defect Size Variations and Their Effect on the Critical Area of VLSI Devices," *IEEE J. of Solid-State Circuits* SC-20(4) pp. 878 - 880 (August 1985).

116. A. Marsh, "Yield Detraction in Simple Photoengraving," *SPIE Proceedings*, pp. 193 - 196 (?).

117. A. V. Ferris-Prabhu, "Modeling the Critical Area in Yield Forecasts," *IEEE J. of Solid-State Circuits* SC-20 pp. 874 - 878 4, (August 1985).

118. J. S. T. Huang and J. M. Daughton, "Yield Optimization in Wafer Scale Circuits with Hierarchical Redundancies," *INTEGRATION, the VLSI journal* 4 pp. 43 - 51 North-Holland, (1986).

119. J. Trilhe, "State of the Art of Wafer Scale Integration," *Proc. of I.F.I.P. Workshop on Wafer Scale Integration*, (17 - 19 March 1986).

120. Carver Mead and Lynn Conway, *Introduction to VLSI Systems*, Addison-Wesley Publishing Company (1980).

121. UW/NW VLSI Consortium, *VLSI Design Tools Reference Manual*, Dept. of Computer Science, University of Washington, Seattle, Washington, Release 2.1 (Oct. 1984).

122. UW/NW VLSI Consortium, *VLSI Design Tools Reference Manual*, Dept. of Computer Science, University of Washington, Seattle, Washington, Release 3.0 (Jan. 1985).

123. B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, Prentice-Hall Inc. (1978).

124. Department of Electrical Engineering, University of Edinburgh, *CLAP User's Guide*. 1985.

125. Tektronix Inc., *DAS 9100 Series Operators Manual*. 1981.

126. Tektronix Inc., "91DVV - The DAS VLSI Verification System," *91DVV1 Operation Manual*, Tektronix Inc. Software Release 1, (April 1986).

127. E. Horowitz and A. Zorat, "The Binary Tree as an Interconnection Network: Applications to Multiprocessor Systems and VLSI," *IEEE Trans. on Computers* C-30(4) pp. 247 - 253 (April 1981).

128. K. M. Kavi, E. W. Banios, and B. D. Shriver, "Message Repository Definitional Facility: An Architectural Model for Interprocess Communication," pp. 271 - 278 in *The 11th Annual Int. Sym. on Computer Architecture*, (June 1984).

129. James Munkres, "Algorithms for the Assignment of Transportation Problems," *J. Soc. Indust. Appl. Math.* 5(1) pp. 32 - 38 (March 1957).

130. Melvin A. Breuer (ed.), *Design Automation of Digital Systems*, Prentice-Hall (1972).

131. Robert Sedgewick, *Algorithms*, Addison Wesley (1983).

132. S. G. Smith, A. Fitzgerald, P. B. Denyer, and D. Renshaw et al., "A Comparison of Micro-DSP and Silicon Compiler Implementations of a Polyphase-network Filter Bank," *Proceedings of ICASSP 86*, pp. 2207 - 2210 (April 1986).

133. J. L. van Meerbergen and F. J. van Wyk, "A 256-point Discrete Fourier Transform Processor Fabricated in a 2μm NMOS Technology," *IEEE Journal of Solid-State Circuits* SC-18 pp. 604 - 609 (1983).

134. G. D. Covert, "A 32 point Monolithic FFT Processor Chip," *Proc. of IEEE Int. Conf. on Acoustic, Speech, and Signal Processing*, pp. 1081 - 1083 (1982).

135. I. R. MacTaggart and M. A. Jack, "A Single Chip Radix-2 FFT Butterfly Architecture Using Parallel Data Distributed Arithmetic," *IEEE Journal of Solid-State Circuits* SC-19(3) pp. 368 - 373 (June 1984).

136. Lawrence R. Rabiner, Bernard Gold, *Theory and Application of Digital Signal Processing*, Prentice-Hall (1975).

137. J. D. Ullman, *Computational Aspect of VLSI*, Computer Science Press (1984).

138. H. S. Stone, "Parallel Processing with the Perfect Shuffle," *IEEE Trans. on Computers* C-20(2) pp. 153 - 161 (Feb. 1971).

139. Per E. Danielsson, "A Variable-Length Shift-Register," *IEEE Trans. on Computers* C-32(11) pp. 1067 - 1069 (November 1983).

140. K. E. Batcher, "Design of a Massively Parallel Processor," *IEEE Trans. on Computers* C-29 pp. 836 - 840 (1980).

141. Bob Mhar, "An Evaluation of a Layout Style Suitable for Automation," *Project Report, MSP 52, Dept. of Electrical Engineering, University of Edinburgh*, (Sept. 1986).

142. H. S. Paul, "Single Phase CMOS and Silicon Compilation," *Project Report, MSP 54, Dept. of Electrical Engineering, University of Edinburgh*, (Sept. 1986).

143. , "Motorola's Sizzling New Signal Processor," *Electronic*, pp. 30 - 33 (March 10, 1986).

144. A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley Publishing Ltd (1974).

145. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman (1979).

APPENDIX A

SOME MATHEMATICAL NOTATION USED IN THE THESIS

A.1. Definition of Big-OH [137]

The expression "f(n) has growth rate $O(n^2)$", or "f(n) is $O(n^2)$" means that the growth rate of the function f(n) is no greater than that of n^2. This does not imply that f(n) is proportional to n^2; it could be less, like nlogn or $\sqrt{n}$. Formally, we say f(n) is O(g(n)) [or f(n) = O(g(n))], if there exist positive constants c and n_0 such that for all $n \geq n_0$, $f(n) \leq cg(n)$. Thus, f(n) may be very much greater than g(n) for small n, but as n gets large and by the time it reaches n_0, it settles down to be at most proportional to g(n); the constant of proportionality is upper bounded by c.

A.2. Permutation

A permutation of n things taken all at a time is an ordered arrangement of all the members of the set. If n is the number of members of the set, the total possible numbers of such permutations is n!, for any one of the set can be put in the first place, any one of the remaining n−1 things in the second place, etc., until all n places are filled.

A permutation of n things taken r at a time is a permutation containing only r members of the set. The number of such permutations is equal to

$$n(n-1)(n-2)\cdots(n+r-1)$$

or $\dfrac{n!}{(n-r)!}$, for any one can be put first, any one of the remaining n−1 second, etc., until r places have been filled.

A.3. Combination

A combination of a set of objects is any selection without regard to the order of elements. The number of combinations of n things, r at a time, is the number of different sets that can be made up from the n things, each set containing r different things and no two sets containing exactly the same r things. This is equal to the number of permutations of the n things, taken r at a time, divided by the number of permutations of r things taken r at a time; that is

$$\frac{n!}{(n-r)!r!}$$

which is denoted by ${}_{n}^{r}C$ in the thesis.

APPENDIX B

PROOF OF NP-COMPLETENESS FOR THE ROUTING PROBLEM

The mathematical proof in this Appendix for the NP-completeness of the message routing problem in a faulty cross-bar switch matrix was contributed by Gordon Brebner of Computer Science Department, University of Edinburgh. His permission has been given to include this appendix in this thesis.

The problem of ***Message routing in a faulty cross bar switch matrix*** (MRFXB) can be defined as follows. Suppose we are given a cross-bar switch matrix X with faults randomly distributed in it. The faults are modelled as either "stuck-on" or "stuck-open" faults†. A set of one-input, multiple-output nets N has to be implemented in such a way that any matrix row accommodates at most one net. Inputs and outputs are located at the top and bottom sides of the matrix, so signals can only enter and leave the matrix via particular columns. In short, MRFXB can be described as follows:

INPUT: Fault pattern for matrix X and net set N.
QUESTION: Can N be implemented by X?

We prove that MRFXB is NP-complete by (a) showing that MRFXB is in the class of NP and (b) that a known NP-complete problem can be reduced to MRFXB in polynomial time, i.e. MRFXB is NP-hard.

Theorem: MRFXB is NP-complete.

Proof:

(a) MRFXB ϵ NP.

† "stuck-on" ("stuck-open") means that the input and output to a switch cell are permanently switched (not switched), so that the on-off state can not be controlled.

A trivial non-deterministic polynomial time algorithm for MRFXB proceeds as follows: (1) Guess an assignment of the nets N to the rows and columns of X, (2) For each net, check whether its input can be switched to the correct output(s).

(b) MRFXB is NP-hard.

The problem SAT which, given a Boolean expression in conjunctive normal form† as input, is to determine whether a variable assignment exists to make the expression true is a well-known NP-complete problem [144,145]. More formally, SAT can be described as:

Input: Set of clauses $c_1, \ldots, c_m$ over variables $x_1, \ldots, x_n$.

Output: Yes if and only if each x_j can be assigned a true-false value such that every c_i is true.

Given an input for SAT, we transform it into a faulty matrix X and a net set N, as follows:

X:	Input Columns	C'_i	for $1 \le i \le m$
		V'_j	for $1 \le j \le n$
	Rows	C_i	for $1 \le i \le m$
		$V_j, \overline{V_j}$	for $1 \le j \le n$
	Output Columns	$D_{ii'}$	for $1 \le i, i' \le m$
		$X_{ij}, \overline{X_{ij}}$	for $1 \le i \le m, 1 \le j \le n$

(see Figure A1) All switches are stuck-open except:

$C'_i \rightarrow C_i$	for $1 \le i \le m$
$C_i \rightarrow D_{ii'}$	for $1 \le i, i' \le m$
$C_i \rightarrow X_{ij}\ (\overline{X_{ij}})$	for $1 \le i \le m$, $x_j\ (\overline{x_j}) \in c_i$

† A Boolean expression is said to be in conjunctive normal form (CNF) if it is the logical product of a set of clauses which are logical sums of laterals, where a literal is either x_j or $\bar{x}_j$. For example, $(x_1 + x_2)(x_2 + \bar{x}_1 + x_3)$ is in CNF, but $x_1x_2 + x_3$ is not.

$V'_j \rightarrow V_j, \overline{V_j}$ for $1 \leq j \leq n$

$V_j (\overline{V_j}) \rightarrow X_{ij} (\overline{X_{ij}})$ for $1 \leq i \leq m, 1 \leq j \leq n$

which are functional.

N: m nets with m + 1 outputs,
and n nets with m outputs

We note that this transformation can be done in polynomial time. The idea behind the construction is that the routability of the first m nets corresponds to the satisfiability of the m clauses, and the routability of the second n nets corresponds to the n variables having consistent values in each clause.

More formally, we show that:

$c_1, \ldots, c_m$ can be satisfied <=> N can be implemented by X.

=> N can be implemented as follows:

For $1 \leq i \leq m$, select a term x_j ($\overline{x_j}$) which makes c_i true and route a net from input C'_i to C_i to outputs X_{ij} ($\overline{X_{ij}}$) and $D_{ii'}$ for $1 \leq i' \leq m$.

For $1 \leq j \leq n$, if x_j is assigned false (true), route a net from input V'_j to V_j ($\overline{V_j}$) to outputs X_{ij} ($\overline{X_{ij}}$) for $1 \leq i \leq m$.

<= First note that each C_i can be switched to more than m outputs and each V_j, $\overline{V_j}$ can be switched to m output. Therefore, the two types of net in N must be realised via the m C_i's and the 2n V_j, $\overline{V_j}$'s respectively.

Further, since each V_j and $\overline{V_j}$ has only one common input V'_j, each of the n m-output nets must use a different j for its column V_j or $\overline{V_j}$, i.e. either V_j or $\overline{V_j}$ is used for every j.

Consider assigning false to x_j if V_j is used, and true to x_j if $\overline{V_j}$ is used. Then, for each i, for some x_j ($\overline{x_j}$) in c_i, the switch $C_i \rightarrow X_{ij}$ ($\overline{X_{ij}}$) must be used, and so $\overline{V_j}$ (V_j) must be used, implying that x_j ($\overline{x_j}$) is true and satisfies clause c_i.

□

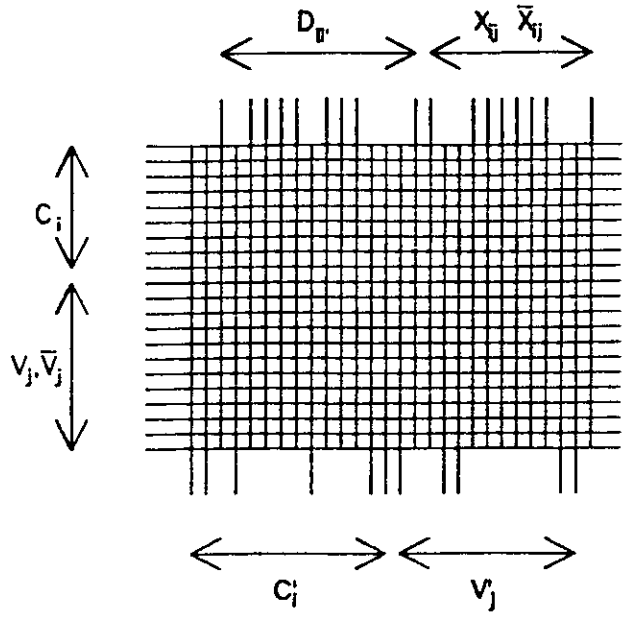

Figure B1 Constructed switch matrix

APPENDIX C

PUBLICATIONS

3.6 FAULT-TOLERANT WAFER SCALE ARCHITECTURES USING LARGE CROSSBAR SWITCH ARRAYS

W Chen, P B Denyer, J Mavor and D Renshaw

INTRODUCTION

The advent of VLSI and the consequent increases in complexity have prompted exciting research problems concerning the reliability, fault tolerance and reconfigurability of integrated systems. Wafer scale integration (WSI), which attempts to reach new levels of circuit integration, offers an opportunity to explore faster computation and real-time signal processing through these three important features. The attraction of WSI for these tasks stems partly from performance advantages that can be achieved by containing such systems within a single, integrated medium, and partly from production economies.

The principal challenge to the attainment of this goal is the development of successful fault-tolerant architectures. Recent research in this area (Koren 1981, Fussell and Varman 1982) concentrated on introducing redundancy into a system which contains faults. Fault-tolerant circuit design using redundancy has made considerable progress in the area of memory design. A few very large memory chips and their design techniques have been announced. (Ueoka *et al* 1984, Kantz 1984, Taner 1984). These achievements are attributable to the architectural regularity of memory which gives itself a variety of efficient fault-tolerant design techniques. The approach is to substitute redundant word or bit lines (address lines) for defective ones during the testing of the chip. The main customisation techniques comprise the use of: latches, laser customisation (Fitzgerald and Thoma 1980, Kuhn 1975), electron beam customisation (Shaver 1984), electrically programmable storage elements (Frohman-Bentchkowsky 1971) and electrically programmable links (Mano 1980).

Latches and other electrically alterable devices are electrically volatile switches which can be reconfigured again during operation, when necessary. Laser and electron beam customised devices are electrically non-volatile switches. Laser customisation is completed by blowing the fuses to good elements, and disconnecting the links to faulty elements. Floating-gate FETs

Proceedings of International Workshop on Wafer Scale Integration

Southampton, England, 10 - 12 July, 1985

(Frohman-Bentchkowsky 1974) are used as electron beam programmable non-volatile switches.

Our approach towards a fault-tolerant architecture for wafer scale integration is to use a large crossbar switch matrix as a reconfigurable interconnect network for serial signals. This section reports research into such an architecture. It is based on extension to work already performed in Edinburgh on bit-serial architecture previously used for the FIRST silicon compiler. In this section, we address in particular a reconfigurable switch array and its potential for fault-tolerant constant-response-time communications.

We will briefly review the FIRST silicon compiler and its architecture in the following sub-section. The wafer scale reconfigurable architecture is presented next. Finally, a summary is given.

FIRST SILICON COMPILER

Bit-serial architectures are distinguished by their communication strategy. Digital signals are transmitted bit sequentially on single wires, as opposed to the simultaneous transmission of words on parallel buses. This distinction is the key to many inherent advantages for bit-serial as a VLSI strategy. Most significantly, bit-serial transmission leads to efficient communication within and between VLSI chips. This is an outstanding advantage where communication issues dominate, as in many signal processing applications.

Within the scope of wafer scale integration or very large integrated systems, fault-tolerant capability is one of the most crucial factors to system performance and economic production. Bit-serial architectures offer good chances of success over their bit-parallel counterparts. Owing to the ease of bit-serial communication relative to parallel, the architectural restriction of 'nearest neighbour' communication need not be imposed, leading to a much greater architectural (and therefore algorithmic) flexibility. Additionally, bit-serial primitives are generally smaller and therefore higher yielding than their bit-parallel counterparts. We have discovered that fault coverage of bit-serial circuits can be high, and may be determined without extensive fault simulation. The cost in silicon, power, complexity and design difficulty is low (Murray *et al* 1983). This leads to a better yield of function per unit silicon area and high fault coverage for the whole system.

Research into bit-serial approaches towards VLSI signal processing in Edinburgh University have resulted in the creation of the FIRST (for Fast Implementation of Real-time Signal Transforms) silicon compiler which has been in use since 1982. The silicon compiler is a tool which takes a high-level functional description as input but produces, instead of machine code like a conventional compiler, a detailed chip mask geometry. FIRST then is simply an interpretation at one technology level of the bit-serial approach.

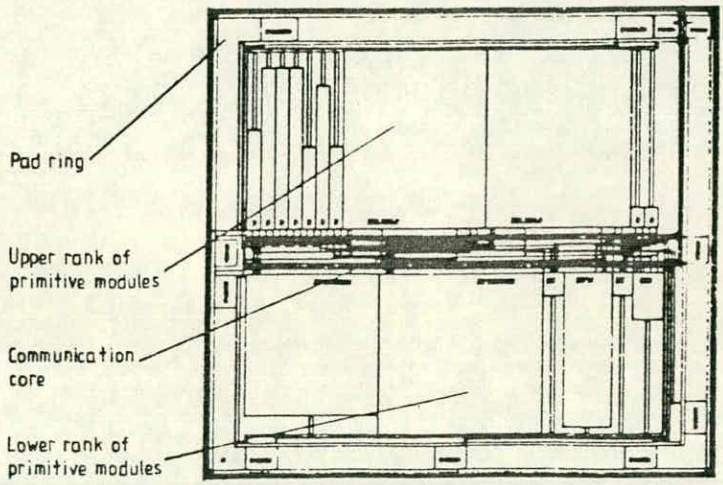

Figure 3.6.1 FIRST floor-plan.

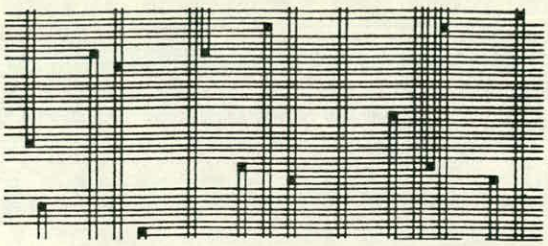

Figure 3.6.2 A wiring channel section.

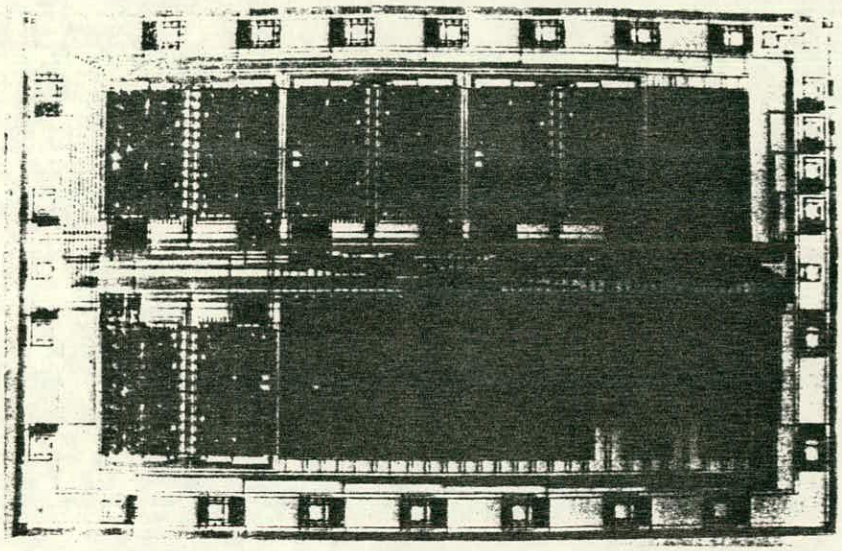

Figure 3.6.3 A FIRST-compiled chip.

Proceedings of International Workshop on Wafer Scale Integration

Southampton, England, 10 - 12 July, 1985

Initially, FIRST-generated chips were targeted at 4–6 micron nMOS technology. This restricted the floorplan style of the chip to that shown in figure 3.6.1. It comprises a central communication channel, flanked by two rows of bit-serial primitive modules. Signal routing is implemented through the central channel only; there is no intimate connection between neighbouring modules. Thus modules communicate by receiving and transmitting data via the channel across the 'waterfront'. Chip input and output signals are routed to peripheral pads via the ends of the channel. A very simple two-layer router is used, with metal lines running horizontally, and diffusion lines running vertically. Figure 3.6.2 shows a section of a typical wiring channel and figure 3.6.3 an actual design generated by FIRST.

WAFER SCALE RECONFIGURABLE ARCHITECTURE

Superchip strategy

A signal processing system created by FIRST is not reconfigurable; instead, algorithms are hard-wired through a central routing channel. The designer can only change the system configuration during compile-time; variations are impossible at run-time. Furthermore, in line with the familiar yield problem, fabrication faults are catastrophic to the entire chip. Finally, the requirement for increasingly large, high-performance, low-cost signal processing systems implies larger chip sizes. Process defect faults limit the practical size of these devices and thus the attainable performance level for one substrate.

The motivation of our research into wafer scale architecture stems from the preceding work on bit-serial architectures for VLSI signal processing, and seeks to overcome the drawbacks of the previous FIRST architecture. The research programme of *superchip* tries to maintain the flexibility to build arbitrarily complex networks of bit-serial processors by concentrating on wafer scale reconfiguration networks for large numbers of bit-serial processors. This is in contrast to the more popular alternative nearest-neighbour-only strategies for wafer scale communication.

Superchip architecture

The superchip is a large bar of silicon containing many independent bit-serial primitives. Computational networks are built with the electrically programmable switch matrix by the end user, who may thus configure this 'standard part' to implement a range of signal processing tasks.

Figure 3.6.4 is a schematic diagram of the superchip. Its architecture is an extension of that used in the FIRST silicon compiler; bit-serial primitives are placed in two rows, above and below a central routing channel. The major difference between this floor-plan scheme and that of FIRST is the substitution of the fixed central routing channel by a large crossbar switch matrix.

Each primitive output drives one horizontal row of the matrix and every primitive input is driven by a column of the switch matrix. Input and output signals are also bit-serial, which enter and leave the superchip via the switch matrix, like all primitive signals. Thus input pads are assigned one horizontal channel in the matrix, and output pads each come from one switch column. The crossbar switch matrix permits users to programme arbitrary functional networks of serial computational elements in real-time.

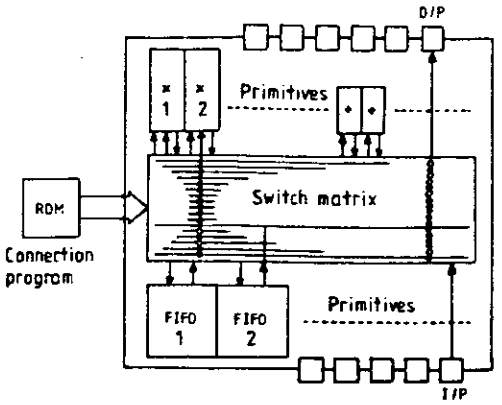

Figure 3.6.4 Wafer scale architecture.

Switch matrix architecture

The following criteria should be considered when a reconfiguration strategy is applied:

(i) it should allow a system to perform its function correctly in the presence of non-functioning modules, e.g. it should provide at least one alternative link between any of the modules in the system,
(ii) it should provide an environment in which the designer may arbitrarily compose a system by using the switch network without fear of message traffic blocking in the network,
(iii) it should provide fault-tolerant capability for the switch network itself,
(iv) the time delay introduced by the network should be kept reasonably low,
(v) because the control circuitry in switch network is usually vulnerable to the defects, it should be as simple as possible; and
(vi) the silicon area overhead should be kept as low as possible.

The size of the crossbar switch array, with its N^2 area dependence, is initially unattractive. However, it exhibits several redeeming features, some of which are better or more suitable in our proposed architecture than other interconnection networks. These features can be summarised as:

(i) full interconnection capability (allowing any connection from one point to any other point),

(ii) non-blocking (compared with some others like banyan network),
(iii) the control circuitry for switching is quite simple; and
(iv) the regularity of the crossbar array allows efficient self-fault-tolerance via redundancy.

Switch matrix design

A key issue in the superchip architecture is the organisation of the switch matrix and its effect on the performance of the whole system. We have considered two different switch cell architectures. In the first of these, we assign one row of the matrix exclusively to each primitive output. Networks are routed in this system by programming connections to the primitive inputs. The alternative switch matrix architecture allows programmable assignment of the horizontal rows for more flexible routing strategies. Figures 3.6.5 and 3.6.6 illustrate the architectures. Figure 3.6.7 shows the circuit diagram of switch cell 1 (SWC-1). It comprises a latch, and a dynamic switch controlled by the latch. When 'column load' is low, the latch is isolated from other circuits. Otherwise the information from the row decoder ('row load' signal) is written into the latch via the pass gate.

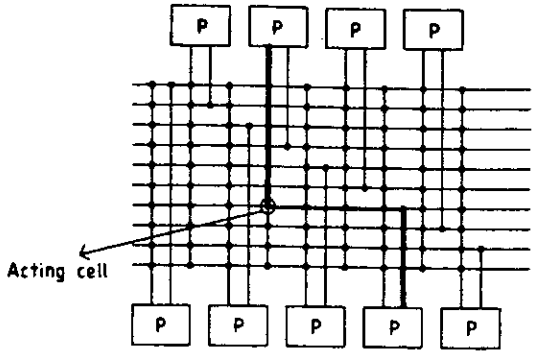

Figure 3.6.5 Dynamic matrix.

Figure 3.6.8 shows the circuit diagram of switch cell 2 (SWC-2). Here the dynamic switch is replaced by a static CMOS transmission gate. Therefore, the performance difference of these two switch cells is characterised by the performance of the CMOS transmission gate against that of the dynamic switch. Suppose that all the transistors in these two parts have the same parameters and q_m is the transconductance for a single transistor. The total conductance of the dynamic switch in SWC-1 becomes $q_m/3$; pre-charging is taken to a whole column of the matrix. In SWC-2, the total conductance of a CMOS transmission gate is equal to $2q_m$; however, because a complete path from a primitive output to a primitive input passes through two transmission gates in series, so the total conductance of a path is equivalent to the conductance of a single transistor. On the other hand, we observe that, in SWC-1, the primitive outputs are not connected to the primitive inputs

directly. A primitive only needs to drive a long horizontal metal track of the matrix (a row of the matrix). In SWC-2, a primitive output has to drive all its horizontal and vertical metal tracks (a row and two columns of the matrix), because the transmission gates connect primitive outputs directly to the primitive inputs they drive.

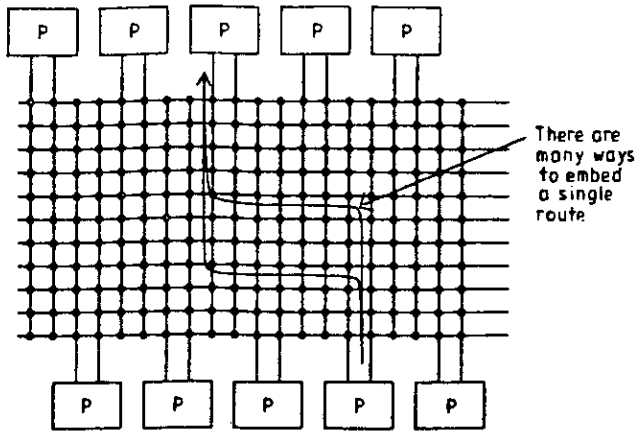

Figure 3.6.6 Static matrix.

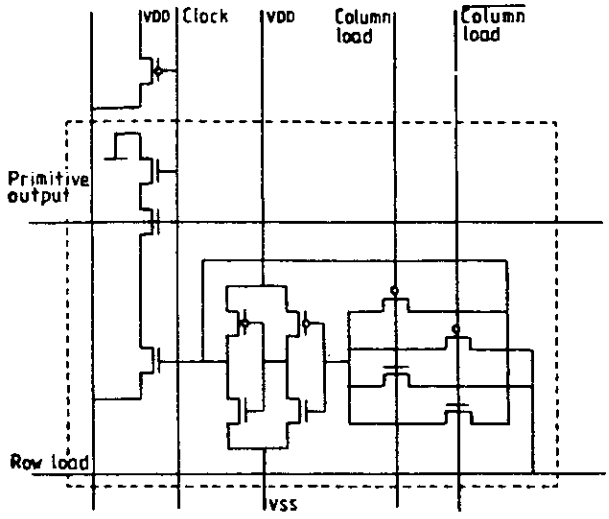

Figure 3.6.7 Circuit diagram of SWC-1.

A trial experiment

Our first test chips comprise a 32 × 32 matrix using SWC-1 and a 32 × 64 matrix using SWC-2 (these have the same interconnection capabilities). We call the first matrix a 'dynamic matrix' and the second a 'static matrix'. Figure 3.6.9 shows the organisation of these two matrices. They have been designed for a 2.5 micron, double metal, CMOS process with a size of

5.2 × 3.1 mm² for the dynamic matrix (figure 3.6.10) and 7 × 2.7 mm² for the static matrix (figure 3.6.11). Using the capacitance parameters of this process, we evaluated that the time delay of a signal passing through both matrices is roughly the same.

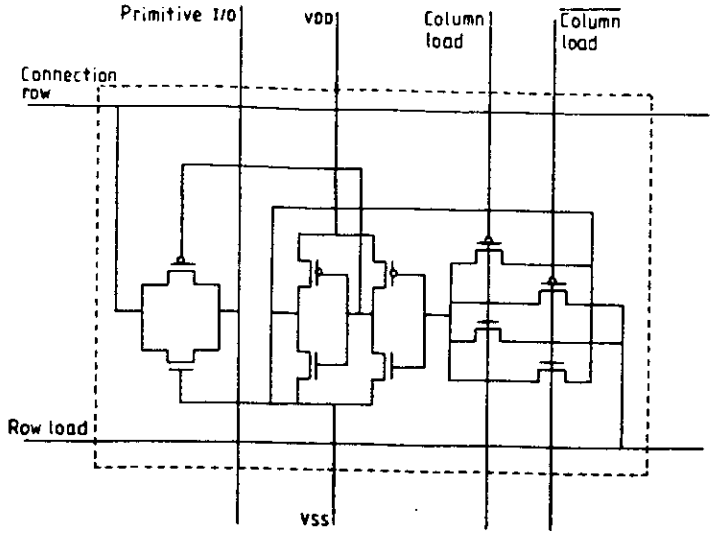

Figure 3.6.8 Circuit diagram of SWC-2.

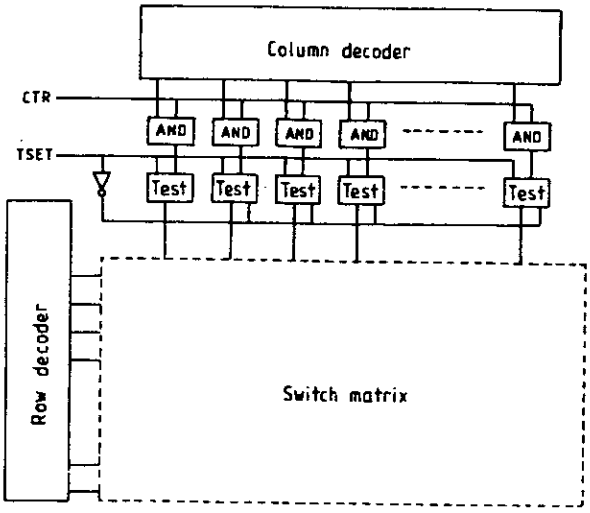

Figure 3.6.9 Trial chip organisation.

Testing and programming

Switch matrix testing is easier than initially anticipated. A faulty switch cell, observed as a switch, can behave in two ways: stuck-open and stuck-on. The

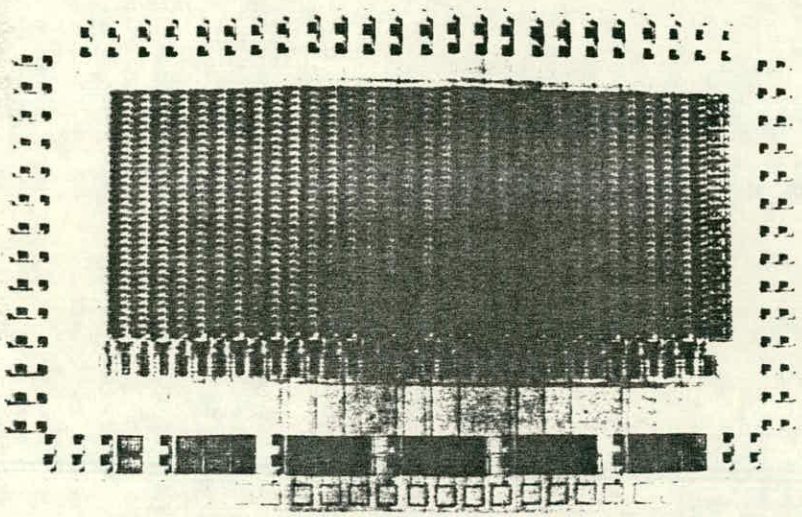

Figure 3.6.10 Dynamic matrix chip.

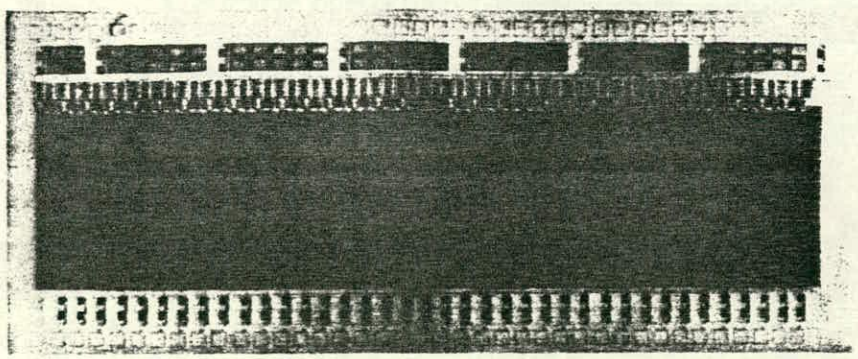

Figure 3.6.11 Static matrix chip.

test set we apply to the matrix is a functional test, the goal of which is to determine whether the switch cell being tested is stuck-open or stuck-on or working normally. Via the signals TSET and CTR, we may set all the outputs from column decoder to 'high' or 'low' at same time. Simultaneously, we activate some row of the matrix; then change the switch cell input of this row to see whether the outputs of the switch cells in this row follow the changes of their inputs. As a result of the testing, every switch cell and primitive is given an error signal $e(i, j)$ as a test signature. An 'initial' signal resets the circuits at the beginning of every test cycle. Suppose that the frequency of the master clock is 20 MHz; it is expected that it will take 800 ns to test a row of the matrix; thus testing a 32 × 32 matrix will need approximately 26 ms of total testing time.

System synthesis within the superchip is invisible to the end user. The whole procedure from the user's system specification to the physical realisation of the system can be divided into three phases as shown in figure 3.6.12. We propose to adopt similar high-level system description language already used in FIRST (Denyer and Renshaw 1985), using a suitable compiler to this high-level language to generate an intermediate code form. According to some criteria of fault tolerance and system performance requirement, the router takes the intermediate form and the test signature of both primitives and switch cells as inputs to produce the routing code for the switch matrix. Using a matching rule for the error signals, the router searches for a possible connection route. If the trial is not successful, the router will back-track to choose another combination for embedding the connections until it finds a complete solution for the system. This algorithm is shown in figure 3.6.13. The last phase is system realisation, which is achieved by loading the routing information into the switch matrix.

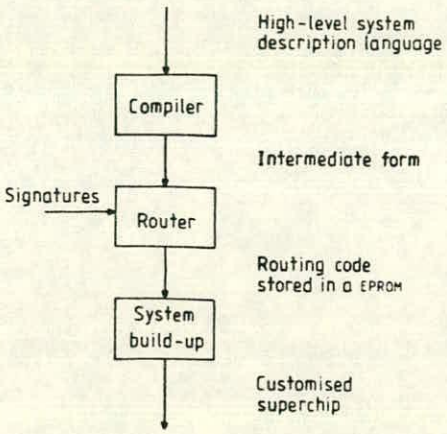

Figure 3.6.12 Superchip programming.

The state of the switch matrix is loaded on power-up, or in response to a reset request from an external EPROM. Although the total state of the matrix is large, the embedded information is much smaller. At most one switch in any column may be active. Thus we need only store and load a row address for the active switch point in each column. Row 0 is reserved for 'no connection' and will be used by unused primitives to inhibit their clock connection. Thus unused primitives need not be active.

This convention leads to the loading arrangement shown in figure 3.6.14. A counter is used to address the external EPROM. Each address is decoded to activate an associated column representing one primitive input line. The active switch for that column is defined by the returned EPROM data generated by the router, which is decoded to drive the relevant 'row load' line.

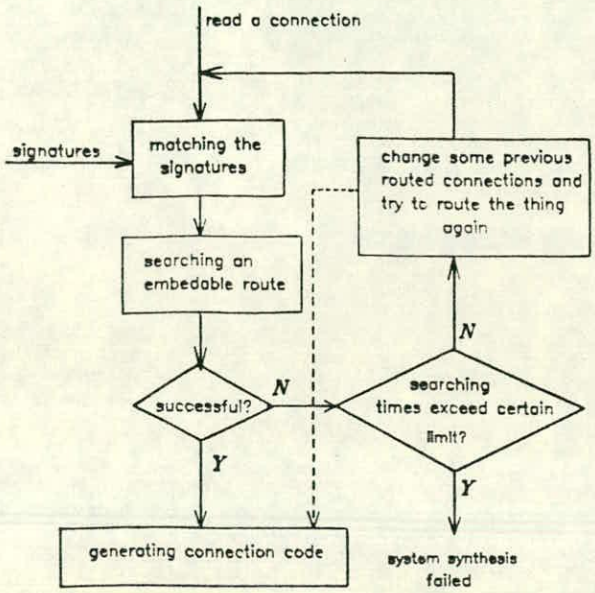

Figure 3.6.13 Routing algorithm.

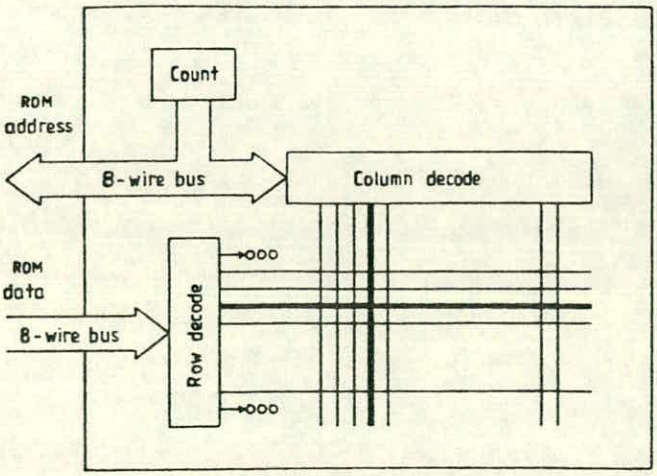

Figure 3.6.14 Loading scheme.

SUMMARY

Modern electronic application areas demand low-cost, powerful, advanced real-time signal processing. This paper presents an architecture for these requirements. Initial indications show that the use of the crossbar switch matrix makes it possible for the user to build a large integrated system economically within a single silicon substrate and to change the configuration easily.

Proceedings of International Workshop on Wafer Scale Integration

Southampton, England, 10 - 12 July, 1985

REFERENCES

Denyer P B and Renshaw D 1985 in *VLSI Signal Processing: a Bit-Serial Approach* (Reading, Mass.: Addison-Wesley)

Fitzgerald B F and Thoma E P 1980 Circuit Implementation of Fusible Redundant Addresses on RAMs for Productivity Enhancement *IBM J. Res. Dev.* **24** 291–298

Frohman–Bentchowsky D 1971 A Fully Decoded Electrically Programmable MOS-ROM *Dig. ISSCC* **14** 80–81

—— 1974 *Solid State Electronics* **17** 517

Fussell D and Varman P 1982 Fault-tolerant Wafer-scale Architectures for VLSI *Proc. 9th Ann. Symp. Computer Architecture* 190–198

Kantz D 1984 A 256 DRAM with Descrambled Redundancy Test Capability *IEEE J. Solid State Circuits* **SC-19** 596–601

Koren I 1981 A Reconfigurable and Fault-tolerant VLSI Multiprocessor Array *Proc. 8th Ann. Symp. Computer Architecture*

Kuhn L 1975 Experimental Study of Laser Formed Connections for LSI Wafer Personalization *IEEE J. Solid State Circuits* **SC-10** 219–228

Mano T 1980 A Fault-tolerant 256K RAM Fabricated with Molybdenum–polysilicon Technology *IEEE J. Solid State Circuits* **SC-15** 865–872

Murray A F, Denyer P B and Renshaw D 1983 Self-testing in Bit-Serial VLSI Parts: High Coverage at Low Cost *Proc. Int. Test Conf.* 260–268

Shaver D C 1984 Electron-beam Customization, Repair and Testing of Wafer-scale Circuits *Solid State Technology* 135–139

Taner R M 1984 Fault-tolerant 256K Memory Design *IEEE Trans. Comput.* **C-33**

Ueoka Y *et al* 1984 A Defect-tolerant Design for Full-Wafer Memory LSI *IEEE J. Solid State Circuits* **SC-19** 596–601