



The Implementation of Testability Strategies in a VLSI Circuit.

John E. Rockliff, B. Sc., B. E. (Hons.)

Department of Electrical and Electronic Engineering
The University of Adelaide
Adelaide, South Australia.

May 1986.

Received 19/5/87

Abstract

The Transform and Filter Brick (TFB) is a 200,000 transistor, CMOS, program controlled, highly interconnected, parallel arithmetic processor for digital signal processing applications.

The chip design has been carried out by a group of postgraduate students of the Department of Electrical and Electronic Engineering at Adelaide University. The architecture and its component modules and control were specified in terms of required function, performance and interfacing by the author in collaboration with Alex Dickinson, and the results are embodied in the TFB system specification document.

As well as maintaining a consultative and coordinational role with regard to the system architecture, the author has been solely responsible for the testability of the chip. A survey of current methodologies, theories and practices in the field of VLSI testing was carried out, with a view to establishing their applicability or otherwise to the design of the TFB chip. A review of this body of literature is included herein.

As a result of this survey, the decision was taken to address the testability problems of the chip using some of these "Design for Test" techniques. Some initial investigations were made into including a totally built-in test system for the control RAM array, but this was put aside in favour of a more global approach to the testing problem, based on scan paths throughout critical areas of the chip. The final system, both the on-chip hardware and the externally applied test program, is specified in this work.

Contents

Abstract.

Acknowledgements. **x**

Declaration. **xi**

Publications. **xii**

1 Introduction. **1**

1.1 The TFB Project. 1

1.2 Implementation Preview. 2

1.3 Testability. 3

2 Faults and Testing. **6**

2.1 Introduction. 6

2.2 Fabrication Defects & Fault Modelling. 8

2.3 Classical Test Generation Approaches. 18

2.3.1 Manual Test Generation. 18

2.3.2 Automatic Test Pattern Generation 24

| | | |
|----------|--|-----------|
| 2.3.3 | Pseudorandom Test Sets. | 28 |
| 2.4 | Design for Testability. | 32 |
| 2.4.1 | Partitioning. | 32 |
| 2.4.2 | Scan Design. | 36 |
| 2.4.3 | Built-In Self Test. | 45 |
| 2.4.3.1 | Test Generation and Application. | 46 |
| 2.4.3.2 | Data Compression Techniques. | 51 |
| 2.4.4 | The Built-In Logic Block Observer. | 58 |
| 2.4.5 | Built-In Self Test Architectures. | 60 |
| 2.5 | Other Test Methodologies. | 61 |
| 2.5.1 | Concurrent Checking. | 61 |
| 2.5.2 | Other Possible Test Techniques. | 63 |
| 2.6 | Summary. | 64 |
| 3 | The Transform and Filter Brick. | 65 |
| 3.1 | Design Overview. | 67 |
| 3.1.1 | Benchmark Tasks. | 67 |
| 3.1.1.1 | The Sum Of Products Computation. | 68 |
| 3.1.1.2 | Fast Fourier Transform. | 68 |
| 3.1.2 | The Logical Architecture. | 69 |
| 3.1.3 | Transformation to the Physical Architecture. | 73 |
| 3.2 | The Data Handling Elements. | 77 |
| 3.2.1 | The Ring Bus. | 77 |
| 3.2.2 | The Data Memories and Pointers. | 78 |
| 3.2.3 | The Arithmetic and Logic Units. | 82 |

| | | |
|----------|--|------------|
| 3.2.4 | The Multiplier/Dividers. | 84 |
| 3.2.5 | The Adder/Subtractor/Shifter/Accumulators. | 87 |
| 3.2.6 | The Input and Output Processors. | 89 |
| 3.2.7 | The "Load Immediate" Bus Register. | 92 |
| 3.3 | Control. | 92 |
| 3.3.1 | General Timing Strategy. | 93 |
| 3.3.2 | Control Hierarchy. | 95 |
| 3.3.3 | The Execution Controller and Microcoding. | 97 |
| 3.3.4 | The Control Store. | 102 |
| 3.4 | Layout and Process Considerations. | 106 |
| 3.5 | Conclusions. | 107 |
| 4 | Testing TFB. | 108 |
| 4.1 | Test Constraints and Aims. | 108 |
| 4.2 | Fault Models. | 112 |
| 4.3 | Test Strategy Overview. | 115 |
| 4.4 | The Control Core. | 121 |
| 4.5 | The ALU and Data Memory Decoders. | 137 |
| 4.6 | The Output Processor. | 139 |
| 4.7 | The Ring Bus. | 143 |
| 4.8 | The Input Processor. | 146 |
| 4.9 | The Data Memories. | 149 |
| 4.10 | The Adder/Subtractor/Shifter/Accumulators. | 151 |
| 4.11 | The Multiplier/Dividers. | 155 |
| 4.12 | The Built-In Test Hardware. | 159 |

| | | |
|----------|---|------------|
| 4.12.1 | The Scan Paths. | 162 |
| 4.12.2 | The Multiplier/Divider Modifications. | 173 |
| 4.13 | Summary of the Complete Chip Test. | 174 |
| 5 | Conclusion. | 176 |
| 5.1 | Test Methods. | 176 |
| 5.1.1 | Design for Test. | 176 |
| 5.1.2 | Functional Testing. | 180 |
| 5.2 | Testing Overheads. | 180 |
| 5.3 | Some General Observations. | 182 |
| 5.4 | Further Work. | 183 |
| 5.5 | Conclusion. | 184 |
| A | TFB Test Specification. | 185 |
| A.1 | The Control Core. | 188 |
| A.1.1 | Start Up Tests. | 188 |
| A.1.2 | Scan Path Verification. | 188 |
| A.1.3 | Write, Wordline Latches | 190 |
| A.1.4 | Main CS Array Decoding, Wordlines, Write Pointer Select. | 190 |
| A.1.5 | BAR Wordlines and Addressing | 191 |
| A.1.6 | CS Column Faults, IPC Incrementer. | 193 |
| A.1.7 | CS Cell Tests. | 195 |
| A.1.8 | Remainder of Control Core. | 198 |
| A.1.9 | Test Times. | 204 |
| A.2 | The ALU and Data Memory Decoders. | 204 |

| | | |
|----------|--|------------|
| A.2.1 | Operation States Decoding. | 205 |
| A.2.2 | LI Disabling. | 205 |
| A.2.3 | LI Selection of Accumulators. | 206 |
| A.2.4 | Test Length. | 207 |
| A.3 | The Output Processor. | 207 |
| A.4 | The Ring Bus. | 212 |
| A.5 | The Input Processor. | 218 |
| A.6 | The Data Memories. | 227 |
| A.7 | The Adder/Subtractor/Shifter/Accumulators. | 233 |
| A.8 | The Multiplier/Dividers. | 243 |
| A.9 | Total Test Length. | 251 |
| B | Multi-Project Chip Test Partitions. | 252 |
| B.1 | The Data Memory and Decoder. | 253 |
| B.2 | The Input and Output Processors. | 254 |
| B.3 | The ALU. | 256 |
| B.4 | The Control Core Elements. | 258 |
| C | Scan Path Yield. | 263 |
| C.1 | Analysis of Yield. | 264 |
| C.1.1 | A Single Long Scan Path. | 265 |
| C.1.2 | Multiplexed Multiple Scan Paths. | 266 |
| C.2 | An Sample Calculation. | 268 |
| D | A Built In Test System for the Control Store. | 270 |
| D.1 | The Test Algorithm and Hardware. | 271 |

D.2 Reasons For Discarding This Design. 280

List of Figures

| | | |
|------|--|----|
| 2.1 | NMOS NOR Gate | 12 |
| 2.2 | CMOS NOR Gate | 14 |
| 2.3 | Charge/Discharge Scheme to Detect Stuck-Open Faults . . | 16 |
| 2.4 | Simple Combinational Circuit: $E = A.B.\bar{C}$ | 21 |
| 2.5 | Trend of Fault Cover vs. Network Size for Practical General Sequential Circuits | 26 |
| 2.6 | Symbolic and Logic Diagrams of LSSD Shift Register Latch (SRL) | 38 |
| 2.7 | Scan Path SRL | 40 |
| 2.8 | Scan/Set Logic | 42 |
| 2.9 | Two forms of linear feedback shift register (LFSR) | 49 |
| 2.10 | Multiple input linear feedback shift register (MISR) | 57 |
| 2.11 | Built-In Logic Block Observer (BILBO) | 59 |
| 3.1 | TFB Logical Architecture. | 70 |
| 3.2 | TFB Physical Architecture. | 75 |
| 3.3 | The Data Memory and Pointers Floorplan. | 81 |
| 3.4 | The Multiplier/Divider Floorplan. | 86 |

| | | |
|-----|---|-----|
| 3.5 | A Floorplan of the ASSA Subsystem. | 88 |
| 3.6 | The Input and Output Processors. | 91 |
| 3.7 | The Execution Controller. | 98 |
| 3.8 | The Control Store. | 103 |
| 4.1 | The Scan Path System. | 165 |
| 4.2 | CMOS Transmission Gate Multiplexor. | 169 |
| 4.3 | CMOS Scan Clock Multiplex/Drive Cell. | 170 |
| 4.4 | CMOS Scan Path Latch. | 171 |
| 4.5 | CMOS Master-Slave Latch. | 172 |
| B.1 | Data Memory MPC Configuration. | 255 |
| B.2 | I/O Processors' MPC Configuration. | 257 |
| B.3 | ALU MPC Configuration. | 259 |
| B.4 | Reduced Control Core MPC Configuration. | 262 |
| D.1 | General Arrangement for CS BIT Scheme. | 273 |
| D.2 | Iterative Flag Generation Circuit. | 274 |
| D.3 | Arrangement of Column Circuits. | 277 |

List of Tables

| | | |
|-----|---|-----|
| 2.1 | Test Sequences for NOR Gate with Open Defect. | 13 |
| 2.2 | Exhaustive Test for Combinational Circuit. | 21 |
| 2.3 | Reduced Test Set. | 22 |
| 3.1 | Data Memory Operations. | 79 |
| 3.2 | ALU Instruction Set. | 83 |
| 3.3 | Input and Output Modes. | 90 |
| 3.4 | Instruction Microcode | 97 |
| 4.1 | The TFB Pin Allocations. | 161 |
| 4.2 | Scan Path Organisation. | 167 |
| D.1 | CS Test Sequence. | 279 |

Acknowledgements.

I would like to thank my supervisor, Dr. Kamran Eshraghian, for his help, advice and boundless enthusiasm, and most of all for instigating this project. I am also deeply grateful to my co-researchers for the experiences and insights gained while working with them. Finally, a special thank you to Steph for helping me through it, and providing a *raison d'être*.

J. E. R.

Declaration.

This thesis has been submitted to the Faculty of Engineering at the University of Adelaide for examination in respect of the Degree of Master of Engineering Science (by research).

This thesis contains no material which has been accepted for the award of any other degree or diploma in any University, and to the best of the author's knowledge and belief contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

The author hereby consents to this thesis being made available for photocopying and for loans as the University deems fitting, should the thesis be accepted for the award of the Degree.

John Rockliff

May 23, 1986

Publications.

K. Eshraghian, A. Dickinson, J. E. Rockliff, G. Zyner, R. E. Bogner,
R. C. Bryant, W. G. Cowley, B. R. Davis, D. S. Fensom,
“A New CMOS VLSI Architecture for Signal Processing”,
Digest VLSI PARC,
Melbourne, May 1984.

K. Eshraghian, R. C. Bryant, A. Dickinson, D. S. Fensom, P. D. Franzon,
M. T. Pope, J. E. Rockliff, G. Zyner,
“The Transform and Filter Brick: A New Architecture for Signal
Processing”,
Proc. VLSI 85,
Japan, October 1985.



Chapter 1

Introduction.

This chapter gives a brief account of the genesis of the TFB project, an outline of the actual circuit, and then gives an introductory consideration to the topic of this thesis, testability of the TFB chip.

1.1 The TFB Project.

The Transforms and Filters Brick (TFB) is an architecture for general purpose digital signal processing developed at the University of Adelaide. The architecture is currently being designed and laid out as a single integrated circuit of around 200,000 transistors, for fabrication using CMOS VLSI technology. The project's main objectives are to examine the effectiveness of the VIVID[†] design environment [108] and to provide a useful tool for digital signal processing tasks, as well as being the medium for several

[†]VIVID is a trademark of the Microelectronics Center of North Carolina.

post-graduate research projects at both Master's and Ph.D levels.

In response to a call from Dr. Kamran Eshraghian for possible large architectures, the local Signal Processing community proposed an architecture for a program controlled, highly interconnected, parallel arithmetic processor (Figure 3.1). The VLSI research group proceeded to examine this architecture with a view to implementation within the framework of the following criteria:

- regularity,
- planarity,
- hierarchical partitioning and
- minimum intercell routing.

In order to realise the original architectural concept, a number of major changes were made to the topology and interconnection of the architecture.

1.2 Implementation Preview.

The processor is intended to perform digital signal processing on 16-bit 2's complement data. The basic data handling elements are four data memories, four multiplier/dividers, four adder/subtractors and the I/O devices. They are connected by a bus structure to allow communication between the elements. The original architecture called for multiple buses with fixed connectivities, but this has been discarded in favour of a single ring-shaped bus. This Ringbus may be switched into discrete segments by pass/break

switch arrays between the data handling modules, or may be used as a single path. This allows a number of local communication tasks to be carried out in parallel, or global communications.

In order to make the design planar without sacrificing parallelism, the multipliers receive one input in parallel from the local bus segment and the other input in serial form from a serial load register connected to a remote bus segment.

The input and output operations are handled by separate processors which control the interface between the pins and the ring-bus.

The program is stored in RAM, and is decoded into control lines which drive the control inputs of the modules. Provision is made for absolute and conditional instructions, including jumps, and wait-on-event instructions. The bulk of the control hardware is physically located in the centre of the chip.

The clocking strategy requires a three-phase non-overlapping clock, primarily to ease the communication problems associated with the highly interconnected parallel processing elements. One phase is reserved primarily for passing information around the ring-bus structure, and for evaluating any conditional program expressions, while the other two are primarily operation cycles within the modules.

1.3 Testability.

The aim of the research work embodied in this thesis is to provide a means or a methodology for testing the fabricated TFB circuit.

From the foregoing sections it can be seen that the TFB chip is a large and complex circuit. The design work is being carried out chiefly by post-graduates of the Department, whose previous circuit design experience is very limited. It should also be borne in mind that the design tools, and the design environment as a whole, are relatively immature, as there has been only a limited amount of circuit design, layout and fabrication carried out using this environment to date. These factors all increase the likelihood of errors occurring in the design of the circuit.

The fabrication process introduces physical defects into the chip in a multitude of ways, and many of these will manifest themselves as faults in the circuit behaviour. Thus it is necessary to test the fabricated chip to ensure the absence of process-related defects.

The assessment of effectiveness of the design environment, and the goal of fabricating working copies of a very useful signal processing tool, requires that distinctions be drawn between these two types of faults. In practical terms this means that a certain amount of fault location must be carried out at the testing stage: it is not sufficient to simply discover that the chip does not work. There must be a reasonable feedback of information regarding prototype failures into the design cycle.

Bearing this in mind, and with due regard to the complexity of the chip, it seemed at the outset sensible to assume that efforts would be required at the design stage to ensure that sufficient information could be derived about chip failures. As it is always easier to design in capabilities at the initial design stage, rather than retro-fitting them, some consideration was given to the concepts of Design for Testability, as well as all of the clas-

sical techniques, and other recently proposed methodologies for assessing information about fault status. These investigations of test techniques are reviewed in Chapter 2.

In order that the reader may appreciate the nature of the testing problems and the proposed solutions, Chapter 3 gives a reasonably detailed review of the actual design of the chip, both at system level and at the module level.

The main thrust of the research, the actual chip test, is detailed in Appendix A, but it is presented there in a highly compact form, and assumes on the part of the reader a great deal of knowledge of the architecture, layout and operational details of TFB. The test sequence is explained in a more coherent fashion, although with less specific detail, in Chapter 4. Some interesting considerations which impinge on the testing problem, but are not central to it, are discussed in the other appendices.

The concluding chapter briefly summarises the results of the chip test design and specification, points out areas requiring further investigation, and makes a few general observations on testability, based on the experience gained in this project.

With the exception of the review chapters (Chapters 2 and 3) and otherwise where specifically acknowledged, all work embodied in this thesis and its appendices, including circuit architecture and design, scan path yield analysis, and chip test sequences, is wholly and solely the author's own.

Chapter 2

Faults and Testing.

2.1 Introduction.

Why do we test chips?

Integrated circuits are designed to carry out specific operations. Early in the design process, a specification is created to describe the intended function of the circuit. At the end of the design and manufacturing processes, we need to know whether the finished product, the integrated circuit, behaves in conformance with that original specification. There are many reasons why it may not.

The design phase may introduce errors into the circuit in many ways: from incorrect interpretation of requirements by the designer, through incorrect logic implementations, incorrect layouts, to incorrect creation of mask-generation tapes. The designer may be at fault, or there may be errors in the computer programs that are used for the computer aided design,

or there may just be a random “soft” error in the electronic storage of the design data.

The designer has at his disposal a plethora of programs to enable him to simulate the behaviour of the circuit, at various levels of abstraction: analogue simulation, timing simulation, switch level simulation, gate level simulation, module level simulation or architectural simulation. The use of these should eliminate most design errors before fabrication [34]. However, particularly where the CAD system is new and unproven, it is unwise to rely on the results of such simulations to assert that a design is correct. The “proof of the pudding” is whether the fabricated integrated circuit actually performs as required.

To ascertain this, we need to physically apply stimuli to the circuit and check whether it responds appropriately. The fabrication process itself inherently introduces imperfections into the physical circuit, many of which have the potential to cause faults in the circuit. Hence a prototype circuit may well be faulty, and the fault may arise from either design errors or fabrication defects. It is important to be able to distinguish between the two at this stage, and this requires the testing to be sufficiently rigorous to locate the fault.

Having established that the design is fault-free, mass production may commence. Any chip may be rendered faulty by a fabrication defect, and so each must be tested to ensure its compliance with the specification. These production tests need not be as thorough as prototype tests, because we know that the design is valid, and if a fault occurs we simply discard that chip, rather than try to establish the exact cause of failure.

The rationale for the emphasis on testing the chip, rather than allowing any faults to manifest themselves during board or system level tests, is simply one of cost. An industry rule-of-thumb [139] is that the cost of testing is exponential on the number of assembly steps completed *i.e.* the test to detect a fault at chip level costs 30c, then a board level test to detect the same fault would cost \$3, a test to find that fault when embedded in a system would cost \$30 and to find that fault in the system in the field would cost \$300.

2.2 Fabrication Defects & Fault Modelling.

During the complicated process of chip fabrication, many factors can introduce flaws or imperfections which may result in circuit faults. These can be broadly grouped into two categories: photolithographic defects and process-quality related defects [80,81,126].

Photolithographic errors result from errors in the pattern generation for mask making, scratching of the mask during usage, dust on the masks, mask misalignment, and wafer warping. Their effect is to cause extra or missing features in the layers of the circuit, or global or local misalignment between the layers. Direct-write electron beam lithography avoids some of these problems, but suffers from some complementary problems: beam enlargement and pattern distortion at the wafer edges, and beam reflection off the stepped edges of previously processed features, writing over incorrect areas.

As well as the various imperfections in the original semiconductor wafer,

such as crystal lattice defects and chemical impurities, process-quality related defects may arise from fluctuations in the operating conditions of process equipment and chemicals. Such essentially random fluctuations affect, amongst other things, metallisation step coverage and contact coverage, oxide quality, and contamination both by chemicals and by dust. The resultant defects include bad quality oxide, pinholing of the oxide, large grainsize metallisation, metallisation lift-off, cracking and shorts, dust particle contamination, chemical contamination, shorts or opens in polysilicon or diffusion interconnect, interlayer shorts, and high contact resistances.

The designer often requires analysis of the effects of possible faults in the circuit. Any gate may have numerous defects associated with it. For instance, one AND gate may have as many as 30 different defects [139]. Clearly modelling a circuit of any useful size to this level of detail is prohibitively complex. Historically the “stuck-at” fault model has been used [139,138,22]. This model assumes that a logic gate either is fault-free, or has an input or output line permanently stuck at one of logic zero or logic one. While not covering some specific types of defects, such as bridging between two signal lines [45,88], the stuck-at model does cover many defects.

If one considers all possible combinations of stuck-at faults within a circuit, the complexity of analysis is still prohibitive for any useful size circuit. A circuit with N nets (lines connecting gate outputs to subsequent gate inputs), where each net can be fault-free, stuck-at-1 or stuck-at-0, has 3^N total possible states. Consider a modest circuit of about 50 2-input gates, containing $N = 100$ nets: the possible number of faulty states of the circuit is $3^N \approx 5 \times 10^{47}$. To analyse each of these states would take

a prohibitive time. Hence industry has utilised the Single Stuck-at Fault model for the generation of tests. In generating the tests, faulty machines are assumed to have exactly one stuck-at fault, with the corollary that any multiple faults in the real circuits under test are assumed to be detected by the single stuck-at fault detecting test set. To date, results have vindicated these assumptions, in that test sets which detect a high proportion of all possible single stuck-at faults in a circuit have resulted in a low number of defective circuits being accepted as good [139,140]. However, analysis by Agarwal and Fung [4] indicates that, whereas these assumptions are valid for circuits without internal fanout, the introduction of reconvergent internal fanout into a circuit dramatically decreases the multiple fault coverage of a single fault detecting test set. This is of some concern, as most circuits utilise internal fanouts. Furthermore, as industry refines its fabrication processes, feature dimensions and spacings are decreasing dramatically, with the consequence that a given size of defect is increasingly likely to cause multiple faults.

As circuit and device dimensions shrink, some second order effects are becoming apparent [80,81]. Incorrect conductor widths can result in unexpected voltage drops along interconnect. Device threshold voltages can be altered by hot electron injection into the oxide, if voltages are excessive. Punchthrough, the merging of source and drain depletion regions, can occur if gate lengths are inadequate. Stored charges can be changed by alpha particle electron-hole pair generation if the layout or critical charge dimensions are inadequate. These are essentially design problems, which can be avoided by careful design and adequate simulation.

Given that a certain circuit is susceptible to the “soft errors” caused by alpha particles, the occurrence of that error is a purely random event in time, and hence this class of fault is not modelled by the stuck-at fault. Shrinking the separation between interconnects can lead to coupling, where a transient on one line can alter the data on the adjacent line. Similarly, the decrease in spacing and stored charge of nodes in densely packed storage arrays can lead to interaction between unrelated nodes, due to charge leakage. These pattern or data sensitive faults are not modelled by the stuck-at fault either.

Other faults occur in MOS technologies which are not modelled by the stuck-at fault [134,55]. These include physical faults in pass gates, tri-state inverters and bi-directional buses[134], and can be considered as faults which lead to a high impedance state rather than a simple logic level, or vice versa. A simple example is given by Mangir [81]. Consider an NMOS implementation of a 2-input NOR gate, as illustrated in Figure 2.1. The fault-free output has two allowable steady states, logic “0” and logic “1”. If an open occurs in the pull-up transistor, then when both inputs are at logic “0” the gate will present a high impedance at the output, rather than the correct output state of logic “1”. A number of physical defects could cause this situation, including

1. an open in the pull-up source or drain, or a missing V_{dd} contact,
2. an open to the pull-up gate, either in the gate polysilicon or in the metallisation of the polysilicon to diffusion contact, or
3. an implant defect which puts the pull-up in a non-conducting state.

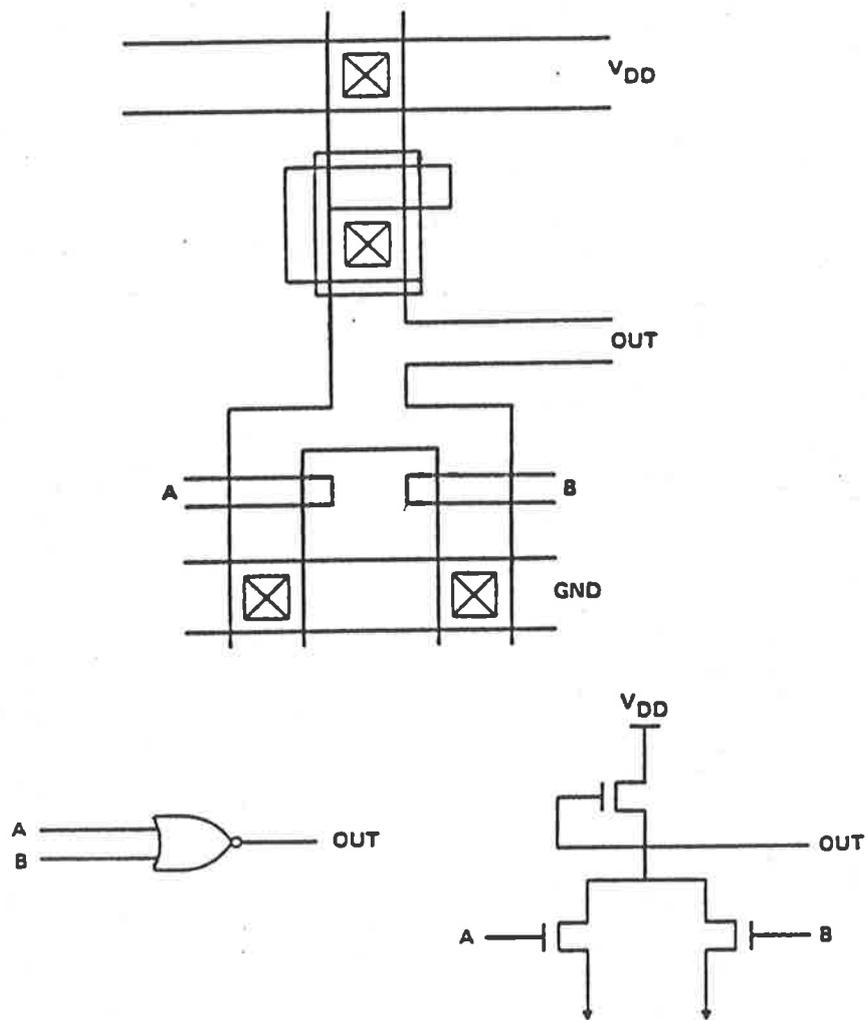


Figure 2.1: NMOS NOR Gate [81].

| Sequence | A | B | $\overline{A+B}$ | F (B4 open) | Comment |
|----------|---|---|------------------|-------------|-----------------|
| 1 | 1 | 0 | 0 | 0 | Fault masked. |
| | 0 | 1 | 0 | 0 | |
| 2 | 0 | 0 | 1 | 1 | Fault detected. |
| | 0 | 1 | 0 | 1 | |
| 3 | 1 | 1 | 0 | 0 | Fault masked. |
| | 0 | 1 | 0 | 0 | |
| 4 | 0 | 0 | 1 | 1 | Fault detected. |
| | 0 | 1 | 0 | 1 | |

Table 2.1: Test Sequences for NOR Gate with Open Defect.

This fault cannot be modelled as a stuck-at fault, because the output is high impedance when the gate exhibits faulty behaviour and may be forced to either logic “0” or “1”, depending on what it is connected to.

Another fault which is of great concern to CMOS users is the “stuck-open” fault [134,102,25,81]. This fault occurs when a gate has an open in a connection to one of the pair of complementary transistors. Consider a 2-input NOR gate [102], as illustrated in Figure 2.2. If, for instance, an open occurs in the connecting line to transistor 4, line B4, then when $B = “1”$, neither transistor 2 nor transistor 4 conducts. Consequently, the input pattern $AB = “01”$ results in the output being in a high impedance state, rather than directly connected to V_{dd} or V_{ss} . Thus, as the output has a certain charge on it as the result of the previous input pattern, this charge will remain trapped at the output upon application of $AB = “01”$. As illustrated in Table 2.1, the response of the faulty gate to $AB = “01”$

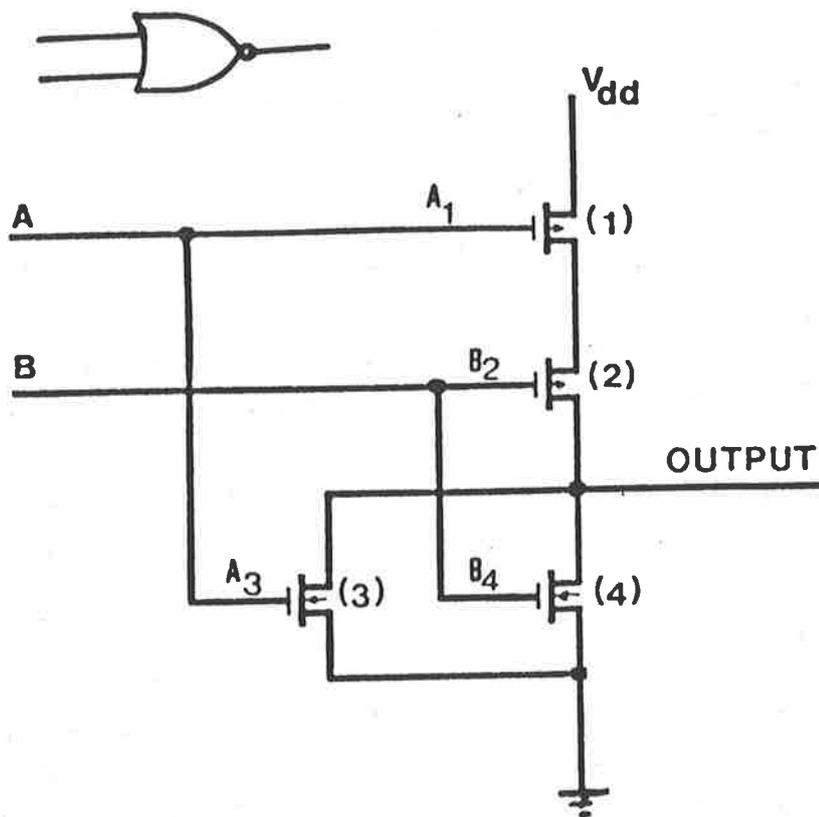


Figure 2.2: CMOS NOR Gate [102].

depends on the response to the previous input. This fault has altered a purely combinational circuit into a sequential circuit, a change that has grave implications on the testing of the circuit. Other examples have been analysed in [134,25]. Generalising, any purely combinational circuit of N inputs may be tested for all possible faults by applying the exhaustive test set of all 2^N possible inputs. However, if that circuit has been changed into a sequential circuit by a fault, then the exhaustive test will not necessarily detect that fault.

Wadsack [134] has proposed a model for CMOS stuck-open faults, which entails the addition of latches on the gate outputs to model charge storage. This model increases the complexity of test pattern generation and simulation to a large extent, and may be impracticable in the general case.

Bozorgui-Nesbat and McCluskey [102] proposed a system whereby an additional transistor is placed at every gate output, allowing the gate to be connected to a charge/discharge (C/D) line by the application of a global control signal (see Figure 2.3).

After each test input, the gate output under test is charged or discharged, forcing it to the opposite logic state to that which it should have currently. If upon disconnection of the C/D line the gate output decays back to its previous (correct) state, it can be said that the gate is not stuck-open for that particular pattern. Conversely, if upon disconnection of the C/D line the output remains at the value forced by the charge or discharge, then a stuck-open fault exists in the gate.

This system seems impractical in the general case because of the high overhead involved. Each gate requires one extra transistor, and more im-

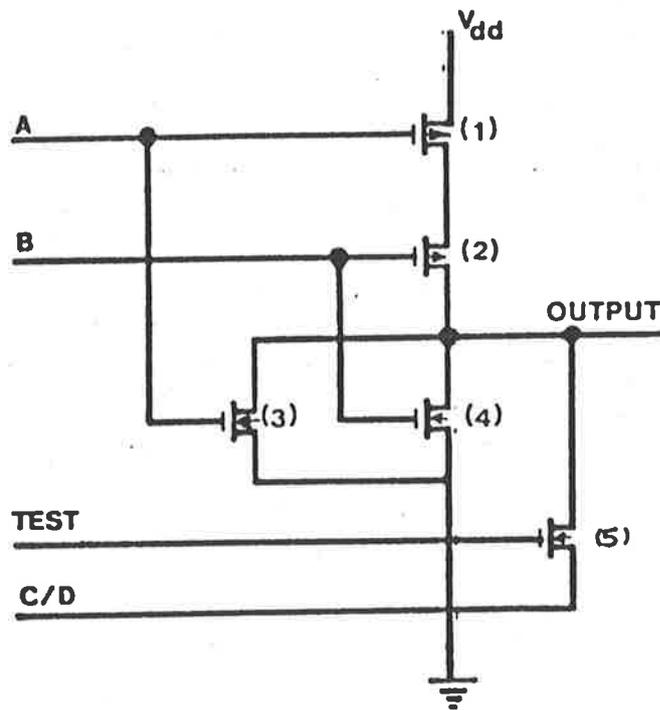


Figure 2.3: Charge/Discharge Scheme to Detect Stuck-Open Faults[102].

portantly connection to both a control line and to the C/D line. The routing problems here are non-trivial. Also, the control signal must be driven above the charge voltage, subjecting portions of the chip to higher than normal operating voltages. Furthermore, since all gate outputs are connected in parallel to the C/D line and are switched by a single global signal, the case must arise where a node is driven to one logic value by its inputs, but the C/D line imposes the opposite logic value. The full V_{dd} to V_{ss} voltage drop occurs across the turned on gate and C/D transistors resulting in static currents and dissipation. Given a sufficiently large circuit, one could say from a probabilistic point of view that for a given input half the nodes are likely to be in the opposite logic state to that of the C/D line, and consequently a large static current will flow in the C/D conductor. Even if this acceptable at device level, with respect to dissipation, the implications of the large currents is that the C/D conductor must be sufficiently large to overcome voltage drop and metal migration problems. The C/D line is required to carry currents of the same order as the main power supply lines around the chip, and since in large systems the power wiring can use major percentages of the routing area to avoid voltage drop and metal migration problems, the addition of an extra C/D conductor of similar proportions may well be impossible. Finally, the concept seems incompatible with the general methodologies of Design for Test, and particularly with Built-In test strategies. Overall, this concept seems to have little merit in the practical case.

More promising results come from Elziq [39], Chandramouli [25], and Chiang and Vranesic [30]. Elziq demonstrates an empirical method for

generating stuck-at fault test sets, although there is some question of the practicality of applying the method to large circuits. Chandramouli utilises analytical methods to show that for networks free of internal fan-outs, all single stuck-open faults may be detected by a specifically ordered single-stuck-at fault detecting test set. For the more realistic case of circuits with irredundant reconvergent fanout, the analysis indicates that limitations on the testability of stuck-open faults are similar in nature to the limitations imposed by circuit topology on the testability of stuck-at faults.

Chiang and Vranesic consider short and open faults in CMOS networks. They show that a test set which detects all open faults in a CMOS combinational circuit will also detect all of a large class of short defects in that circuit. It is shown that ordering of the test sequence is essential for the detection of open faults, and a heuristic method is suggested to carry out that ordering.

Although the CMOS stuck-open fault has been known for some time, and methods suggested to overcome the associated problems, there is still no general agreement on how widespread or common these problems are, and any solutions to these problems are still moot.

2.3 Classical Test Generation Approaches.

2.3.1 Manual Test Generation.

Manual test generation describes the process whereby a designer or test engineer specifies a test set for a circuit without using algorithmic or com-

putational tools.

The situation often occurs that a circuit must be tested, but the test engineer knows little or nothing about the internal structure of the circuit. This is particularly the case for system manufacturers whose products incorporate complex chips fabricated by another company *e.g.* microprocessors. The test engineer usually knows the instruction set and other data which specify the normal behaviour or function of the circuit, but the details of the internal structure of the circuit are generally unavailable. Hence the tendency is to test the circuit for correct execution of its normal functions. This is known as functional or behavioural testing.

The problem arises that to ensure that the circuit will work correctly in all cases, it must be tested for all possible states, including all possible data states, since no information is available on the internal structure and hence no interactions can be discounted. This leads to impossibly long test sets. One approach to this problem is to heuristically select which data inputs are likely to detect certain fault types *e.g.* inputs to set up a maximum length ripple carry in an adder module. Note that even here the simplification comes as a direct result of knowing or assuming some limited information about the internal structure of the circuit. The usual format for such test is to evaluate the circuit's response to its normal commands operating on a limited subset of data inputs (*e.g.* [9,141]). Typically this approach has resulted in low fault coverage, which means that a large portion of faults pass through the test undetected [139].

However, there has been a considerable body of research into this problem (*e.g.* [89,3,7,1,71,12,13,129,18]). Much of it has been based on the

approach of modelling the unknown internal structure of the circuit using register transfer language [3,89], or other abstracted descriptions such as directed graphs [7,18]. This has had some promising results, with some researchers achieving fault coverage of over 90% as a result of test generation on the abstract model [71,3]. Many have also indicated that the test generation has been, or can easily be, automated [13,12,1,18].

The use of functional or behavioural testing is certainly applicable to the chip buyer as an acceptance test, even on the basis that any testing is better than none. In most cases, for the manufacturer of the chip, the functional test is not adequate assurance of the chip's performance, and given that the design information is available, far better test sets are possible, in terms of both fault coverage and test costs. (An example of a manufacturer using a functional test is given by Lindbloom *et al.* in [78].)

There exist certain forms of circuit that are completely tested by specific types of test set. The designer knows that if he adheres to all the constraints for that class of circuit, then a predetermined "universal" test set may be used. The classic example of this is the class of purely combinational circuits. These may be tested by application of all possible input combinations: this is known as an exhaustive test set. An N input purely combinational circuit requires 2^N test vectors. A simple combinational circuit is illustrated in Figure 2.4, and its exhaustive test set is tabulated in Table 2.2.

While this is a relatively trivial example, other circuit classes can be made testable by universal test sets at the cost of extra design overhead. One class which has received considerable attention



Figure 2.4: Simple Combinational Circuit: $E = A.B.\overline{C}$

| Test No. | Node Name | | | | |
|----------|-----------|---|---|---|---|
| | A | B | C | D | E |
| 1 | 1 | 1 | 0 | 0 | 1 |
| 2 | 1 | 0 | 0 | 1 | 0 |
| 3 | 0 | 1 | 0 | 1 | 0 |
| 4 | 0 | 0 | 0 | 1 | 0 |
| 5 | 1 | 1 | 1 | 0 | 0 |
| 6 | 1 | 0 | 1 | 1 | 0 |
| 7 | 0 | 1 | 1 | 1 | 0 |
| 8 | 0 | 0 | 1 | 1 | 0 |

Table 2.2: Exhaustive Test for Combinational Circuit.

| Test No. | Node Name | | | | |
|----------|-----------|-----|-----|-----|-----|
| | A | B | C | D | E |
| 1 | SA0 | SA0 | SA1 | SA1 | SA0 |
| 8 | SA1 | SA1 | SA0 | SA0 | SA1 |

Table 2.3: Reduced Test Set.

[51,112,67,111,53,66,132,49] is that of Programmable Logic Arrays (PLAs). The rationale for this is that PLAs are very sparse, in that for a given number of input variables they implement only a small number of minterms. This means that though they are testable using exhaustive test sets, these are in practice very inefficient. On the other hand, the universal test sets typically grow linearly with increases in the inputs, rather than exponentially.

As mentioned previously, an exhaustive test set can be guaranteed to test a purely combinational circuit. However, it may not be the shortest test set to completely test the circuit. Consider the circuit above, Figure 2.4, and its exhaustive test set Figure 2.2. If we are interested in detecting all single stuck-at faults in this circuit, then the test set consisting of Test Numbers 1 and 8 will suffice, as illustrated in Table 2.3. Extending this argument to a circuit with say 20 inputs, the exhaustive test set would consist of $2^{20} \approx 10^6$ test vectors. This exponential growth of the test set leads directly to increased test times and hence costs. Thus there is considerable incentive to seek a smaller test set, and as was illustrated in the example, this can be found in many cases if knowledge of the detailed internal structure is available.

While exhaustive testing is useful for combinational circuits up to a cer-

tain size, the introduction of latches quickly increases the test size. Since each single bit latch can be set to one of two states, a circuit of N inputs and M latches has 2^{N+M} possible states, each requiring a test vector. Furthermore, setting up the internal latches to the required state requires further input sequences. Thus the exhaustive test set for a sequential circuit grows faster than exponentially on the sum of inputs and latches. For example, a modest LSI circuit of 25 inputs and 50 internal latches would require a total of $2^{75} \approx 3.8 \times 10^{22}$ test patterns, excluding the set-up patterns. If these could be applied at the rate of ten every microsecond, the test would take over 10^8 years - somewhat longer than the average product lifetime!

A test which utilises knowledge of the circuit's internal construction is termed a "structural" test. Traditionally the structural knowledge has been expressed as a gate-level model of the circuit, but this has been mainly as a convenience, particularly as many designers still conceptualise in terms of gates, rather than in structures such as pass transistor steering logic or domino logic.

The essence of structural testing is to postulate all fault conditions which could occur under the structural model, and for each one attempt to find a suitable input test vector, or sequence of test vectors, to cause that condition to be detected. An extension to this scheme is to analyse which other fault conditions are detected by the input vector that detects the original postulated fault. If the circuit to be tested is not too complex, the generation of a structural test set may be done manually. The circuit complexity is determined in part by its size, regularity, and partitioning.

The canonical examples of large circuits with heuristic structural test

sets are Random Access Memory (RAM) chips. Here the regularity of the design enables the chip test to be considered essentially as the test for a single memory cell, replicated the correct number of times. Due to the importance of bulk memory in computing, and the effects of shrinking device technologies on large RAM chips, a large amount of research has been done into optimum test patterns for RAM. Though many test sets of varying complexity have already been described (*e.g.* [65,22]), the continuing evolution of the fabrication processes and chip architectures compels ongoing research [99,123,145,122,72,64].

2.3.2 Automatic Test Pattern Generation

In practice, most digital circuits are neither small enough nor regular enough to allow manual generation of a structured test set. This has led to the use of algorithmic techniques implemented as computer programs, allowing automatic test pattern generation (ATPG).

The D-algorithm for test generation was described by Roth in 1966 [109], and was quickly taken up as one of the prime methods of generating test sets automatically. The algorithm generally requires a gate-level description of the circuit as input, and proceeds by locating paths from primary inputs and outputs to specific faulted nodes. Other algorithms that also utilise this “principle of paths” are PODEM (Path Oriented DEcision Making), first described by Goel in 1980 [57], and FAN, described by Fujiwara and Shimono [52] in 1983. These are all algorithms in the strict sense that if a test exists for a fault, the algorithms will generate it [92]. Other methods

have been proposed for which this is not true, and the user is then in the situation of not knowing whether the program simply failed to generate a test or if the fault is untestable.

These algorithmic methods have certain limitations in practice. First is that, although both algorithms are commonly used to generate test sets for sequential networks, they are actually designed for use on combinational networks only. Whereas their performance with regards to fault cover is good for combinational circuits, remaining adequate for networks in excess of 5000 gates [139], degradation of fault cover occurs due to the increased complexity of sequential elements. Figure 2.5 from [139] illustrates this, showing that for general sequential networks with gate counts in excess of 2000, the fault cover obtained by algorithmic methods is unacceptably low. The practical size limit for algorithmic test generation for sequential circuits is generally considered to be around 5000 gates [58,144] if the logic is structured [37], and less for purely random logic.

Historically this fault cover fall-off has been attacked by gaining access to critical nodes within the circuit, to observe results at and/or control the nodes. The classical method of mechanically probing test points at nodes within the circuit to improve controllability and observability is no longer applicable, due to the ever-decreasing dimensions of the circuit features. Although non-intrusive probing has been achieved using a scanning electron microscope with voltage contrast imaging [79], this type of system appears to be limited to prototype testing, due to its low test throughput. Given the increasing density of VLSI circuits, and the tendency for the packages to be pin-limited, the scheme of bringing out internal nodes to dedicated

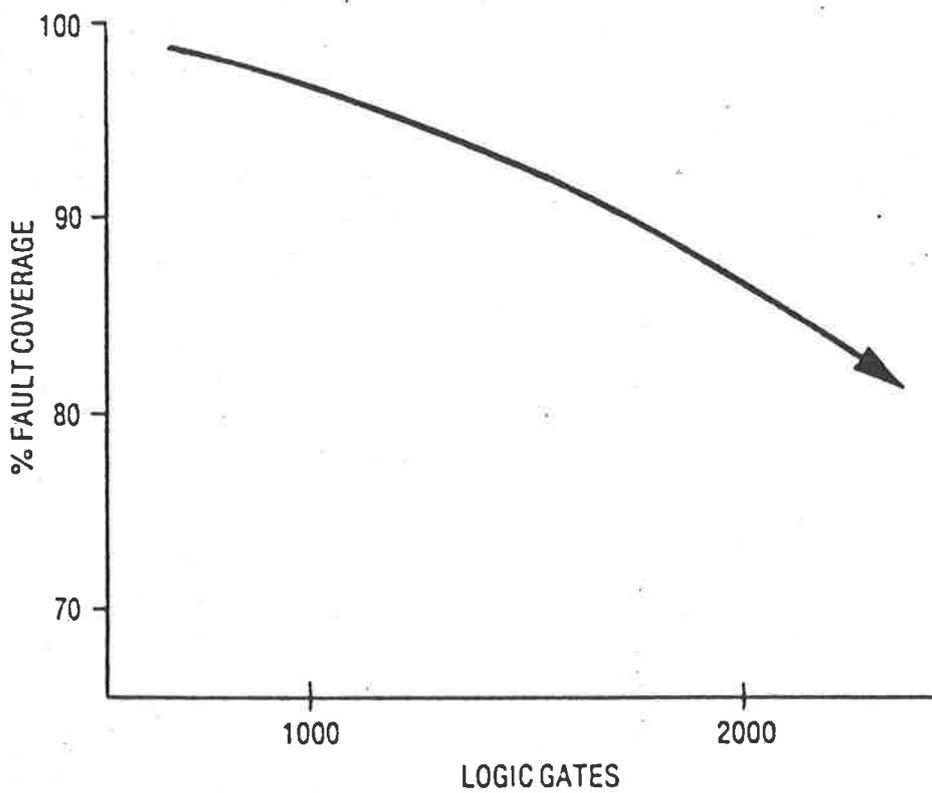


Figure 2.5: Trend of Fault Cover vs. Network Size for Practical General Sequential Circuits [139].

pins does not seem to have wide applicability either.

Some test generation systems have been created which rely not on one single technique, but on several [143,58,17]. A typical example is PODEM-X [58], which uses a combination of the test sets generated by a Shift-Register Test Generator (SRTG), a Random Path Sensitisation Test Generator (RAPS) and the previously mentioned PODEM algorithm. The SRTG provides patterns to test the shift-register scan paths (see Section 2.4.2 for further discussion of scan paths), while RAPS generates tests using a random selection of sensitised paths. In PODEM-X, dynamic assessment is made of the performance of the generators at certain intervals, and this information is used to select which of the three to apply. Other systems simply apply the SRTG and random test generators to generate tests for all the "easily testable" faults, and then apply an algorithmic generator to the "hard to test" faults to either generate a test or confirm the untestability of the fault.

The use of these "mixed mode" systems has the effect of increasing the useful span of generation systems up to circuits of the order of 50,000 gates, provided that the logic is structured in a suitable manner [37]. For instance, Goel and Rosales in [58] cite examples of circuits of 33,500 and 48,600 "equivalent" gates (*i.e.* the logical models of the circuits contained that many gates), with respectively 90,000 and 134,000 single stuck-at faults, for which the PODEM-X system generated test sets of 3213 and 5150 input vectors, with fault coverage of 88.3% and 88.5% respectively.

This however brings us to the second problem, that of increasing test generation times and the concomitant increase in test cost. The above

examples from [58] were achieved at the expense of approximately 8 hours and 23 hours of CPU time respectively on an IBM 370/168, and these times are purely for the generation of the test sets. The next step in the process is the simulation of those test sets, which involves not only simulating the response of the "good machine" to get the reference response to each input, but also simulating the response of each "faulted machine" to the complete test set. Hence if a circuit has N possible faults, the simulation phase consists of applying the complete generated test set to $N + 1$ machines, consisting of N faulted and 1 good simulation. It has been observed in practice that the computer run time required for simulation alone is proportional to n^2 , where n is the number of gates, and when the generation phase is taken into account the run time becomes proportional to n^3 (see [140] for discussion of this point).

2.3.3 Pseudorandom Test Sets.

One solution to the difficulty of deterministic test vector generation is to simply apply pseudorandom input vectors *i.e.* algorithmically generated vectors satisfying randomness properties and with appropriate statistical distributions. The problem then arises of determining exactly what faults are detected by a given length random sequence, or conversely what length random sequence ensures adequate fault coverage. The traditional method is to perform fault simulations for the random test set, with consequent high CPU cost:

This high cost of simulation has given impetus to the search for efficient

testability analyses - programs that operate on the description of a logic network to predict the fault cover achievable by a random test set of given size, and to isolate those circuit areas which require modification to improve their testability. Such programs have been in existence for some years, the best known being SCOAP, the Sandia Controllability/Observability Analysis Program [59], and Camelot, a Computer-Aided MEasure for LOGic Testability [14]. These programs use rules to assign "weights" to node controllability and observability in a gate-level circuit description. It has subsequently been shown [5] that the correlation between a fault's weighting and its probability of detection by random patterns may not be high, and thus some care must be taken in the analysis of random pattern fault cover using these programs.

All such algorithms are made inaccurate by reconvergent fan-out in a circuit, and this has hampered efforts to use random pattern generation. Savir, Bardell and Ditlow in 1983 published the "fan-cutting" algorithm [116,117], which overcomes this problem by "cutting" the fan-outs. The algorithm gives proven correct upper and lower bounds on the probability of detecting a stuck-at fault in a network, the only difficulty being that in certain cases the lower bound for detection probability may be zero, which makes it difficult to assign an upper bound to the number of test vectors required (without recourse to fault simulation).

Recent work by Jain and Agrawal [70] and Brglez [19,20,21] indicates the possibility of a solution of the above-mentioned problems. Jain and Agrawal have published a STATistical Fault ANalysis technique (STAFAN), which is essentially the application of a set of random input vectors to

an augmented gate level fault-free simulation. The signal statistics of the simulation are accumulated by counters assigned to each node, and these are interpreted as approximate fault detection probabilities. STAFAN results for circuits ranging from 102 gates in size to 2723 gates were given, along with the results of gate level fault simulation, and these showed very good agreement between the two fault cover figures. The major points in favour of this method are that:

1. it is a simple augmentation of the fault-free simulation, which is required regardless of test generation,
2. the CPU overhead involved in augmenting a simulator to execute STAFAN is linear on the number of gates, and
3. the method has procedures for handling feedback, which allows it to derive fault cover estimates for sequential circuits.

Brglez has published a controllability/observability program COP which implements a recently published testability analysis algorithm [19,20]. This algorithm utilises testability measures which, while similar to those of Jain and Agrawal, vary subtly in definition. The algorithm calculates the controllability and observability values for the circuit nodes from the gate level description in one forward pass and one reverse pass through the network. No simulation, either fault-free or faulted, is actually required to derive these figures. The method has demonstrated linear growth of CPU costs on the gate count in all examples tried to date.

In [21], Brglez outlines ways in which the COP program can be extended, to providing heuristic information to ATPG algorithms such as

FAN to increase their efficiency, to providing partitioning data, and to predicting fault cover both with and without the application of test sets (the former is very similar to STAFAN). Examples of fault cover prediction are given for a number of circuits, varying in complexity from 206 lines to 3640 lines, comparing results from both methods with the results from a concurrent fault simulator. All these circuits embodied reconvergent fan-out, and one example had redundant faults. The results indicate a very good agreement between the COP fault cover predictions and the actual fault cover, with the method using actual applied patterns showing marginally better results at the cost of more calculations. Note that while COP is designed for purely combinational circuits, Brglez comments that the similarity to STAFAN is such that by the incorporation of the feedback handling technique used in STAFAN into COP, sequential circuits could also be analysed.

The latter two methods are relatively recent results, and their effectiveness cannot be judged fully as yet. However, the promise is there of cheap test set fault cover prediction, and it would be very surprising if this does not accelerate the movement away from expensive deterministic automatic test pattern generation towards pseudorandom test sets. Even so, the test generation costs will still rise with increasing circuit complexity. Also, it is well known that certain circuit types are resistant to random testing, in that they either require very long test sets to ensure adequate fault cover or circuit modifications are required to enhance controllability and observability. Typical examples are PLAs with their high fan-in, and other circuits with high fan-in (example in [69]). This latter point justifies continued research into other forms of test generation.

With the trend from LSI to VLSI, the typical part production runs have decreased in quantity, particularly for semi- and full-custom chips. Thus the once-off costs of test generation, plus the capital costs of the more advanced testers required, have to be amortised against a smaller number of products. This, coupled with the increase in difficulty and cost of test generation, has meant that the chip production cost is increasingly dominated by the testing costs. Clearly, as the density of integration and overall gate count of circuits increases, automatic test generation systems become less effective in minimising test costs, and hence the attention of the designer must focus on methods of making the circuits testable. These methods are collectively known as "Design for Testability".

2.4 Design for Testability.

The term "Design for Testability" is used to describe a variety of techniques of incorporating logic structures into the original circuit design to facilitate testing [139,140,85,86,82,92].

2.4.1 Partitioning.

Given that current automatic test pattern generators are capable of performing adequately on circuits up to a certain size, one obvious approach to the problems of testing larger circuits is to partition them into smaller subcircuits which can be tested independently of each other, and are small enough to allow automatic test generation. This requires external controllability of the inputs and external observability of the outputs of each sub-

circuit or partition. In fact, if the partitions can be made sufficiently small, then other test generation techniques such as exhaustive testing become viable again, with the consequent advantage of little or no fault simulation being required [84].

The problem of ensuring that the circuit will be partitionable for testing is best considered as an architectural constraint. Certain types of system are amenable to easy decomposition into separate partitions. The most obvious of these is when a circuit is composed of two or more discrete non-interacting subcircuits. This is unusual in practice, but there are benefits from minimising the interaction between subcircuits. In particular, two forms of system architecture are recognised as aiding the testability problem: bus-oriented architectures and bit-sliced architectures.

Bus-oriented architectures have gained a wide acceptance in microprocessor systems. Their predominant advantage in testability arises from the fact that most of the critical control information and data is communicated from module to module via the system buses, with very little other inter-module communication, and those system buses are generally available, if not externally, at least for testing purposes. One problem associated with this type of architecture is that the separate bus modules can be so complex that they are not adequately small enough partitions with respect to testing, and require further partitioning or other design enhancements to ensure their testability (*e.g.* [31], and Section 4.11 on TFB multiplier). Another drawback is that the buses themselves are liable to failure, and the cause may be any one of the connected modules, or the actual bus hardware. Whereas normal testing is carried out by deducing fault location

from the voltage response, bus fault location may need the more difficult procedure of current measurement [140].

Bit-sliced architectures have also gained a wide acceptance, particularly in the wake of the Mead and Conway inspired trend to regular structures [87]. They can be considered as a subgroup of one-dimensional iterative logic arrays. It has been shown that one-dimensional iterative logic arrays can be made testable by the adherence to certain conditions concerning controllability and observability [104], and applying further conditions can make the test set size independent of the array size (*i.e.* processor word size) and identical for each array element (*i.e.* each bit-slice) [124].

The foregoing schemes require a high level of external access to the partition boundaries. In some cases, distinct partitions within a circuit are obvious, but there is a problem in gaining external controllability and/or observability of the nodes at the internal partition boundaries. Several solutions have been suggested to overcome this problem simply. The first is to directly wire out the nodes in question to the pins. In view of the tendency of VLSI circuits to be pin-limited, it may be preferable to multiplex the internal node lines in a test mode to pins used by the circuit inputs and outputs in its normal operational mode. The disadvantages of these schemes are the increased capacitive loading and consequent performance degradation at the internal nodes, the areal penalty of the extra routing and in the second case the decreased reliability and increased delays of the normal circuit inputs and outputs due to the addition of the multiplexor cells. Gepraegs and Tandjung [56] suggest an extension to this scheme to remove the performance effects of the extra test wires on the internal nodes

by removing them from circuit. They suggest that the wires be fused open by passing a high current through the extra conductors, but it would appear that the laser-fusible link technology developed for reconfiguring redundant circuits is far more reliable and predictable.

Of course, some circuits exist in which the internal partition boundary nodes inherently have adequate controllability and observability. Examples are given in [101,102] and [82, pages 15-16]. Paths from the circuit inputs to the subcircuit inputs, and from the subcircuit outputs to the circuit outputs are *sensitised* by applying the appropriate test pattern to the circuit inputs. These paths are used to propagate the test set and test results to and from the internal partition. Such a partition is known as a sensitised partition. Many circuit structures are amenable to such partitioning by inspection, rather than by automated algorithm.

Partitioning can be done on the basis of back-tracing from circuit outputs to primary circuit inputs. Some automatic partitioning tools exist, but they do not necessarily give totally disjoint partitions, so that the resultant partitions have certain inputs and/or outputs which are embedded in other partitions. Furthermore, the partitioning problem has been shown to be NP-complete in general, with the consequence that the partitioning calculations are very costly in CPU time [69]. To summarise, although partitioning aids solutions to the testability problems of large circuits, it is not always a complete solution in its own right.

2.4.2 Scan Design.

As discussed previously, while combinational logic is relatively straight forward to test, the addition of sequential elements can lead to severe degradation of test fault coverage. If the sequential elements can be set to any desired state, and their contents examined at any time, then the problem of testing circuits containing sequential elements reduces to that of testing those subcircuits, consisting purely of combinational elements, which lie between inputs and sequential elements, between sequential elements and outputs, and between separate sequential elements.

The method known generically as Scan Design proposes that the memory elements of sequential circuits are linked together to form shift registers which originate and terminate at external pins. Upon application of certain control signals, the current values held in the circuit's memory elements are shifted out, and new values may be shifted in, providing an opportunity to alter the circuit state. The effect of providing such controllability and observability at internal nodes is essentially to partition the network into smaller subcircuits of a purely combinational nature, with the consequent benefits of increased ease of test generation and shorter test times. It should be noted that the Scan Design specifically excludes the memory elements of embedded RAM arrays, primarily because their array architecture and the memory cell designs make them susceptible to faults which cannot be tested under the Scan Design regime. These embedded arrays require special treatment to enhance their testability, as discussed by Eichelberger *et al.* [36], Westcott [135] and Sun and Wang [130].

First suggested by M. Williams in 1973 [137], there are now several common variants on the Scan Design scheme which differ in implementation details, as well as a related scheme, Random Access Scan. The Scan Design variants are Level Sensitive Scan Design (LSSD), Scan Path and Scan/Set.

LSSD is the structured design methodology adopted by IBM. Described by Eichelberger and Williams in 1977 [37], the salient features of this technique are the requirements for master-slave latch pairs, two non-overlapping clocks for the scan function and the requirement that the shift register latches (SRLs) be level sensitive. The requirement for master-slave latch pairs is to avoid information loss on shifting, and is common to all three Scan variants. The term level-sensitive implies constraints on system latch design and clocking to ensure that the latches respond to levels rather than edges, and to make them relatively immune to the ac characteristics of the clocks. The two-phase clocking is specified to avoid race conditions caused by the feed-back of the slave latch output into logic gating the data input to the master latch. For a more complete description of the circuit and clocking restrictions, refer to [37]. Chen in [27,28,29] derives general rules for the avoidance of race and hazard in scan structures, and shows that the LSSD design rules are a simple but sufficient subset of these general constraints.

A typical LSSD shift register latch is illustrated in Figure 2.6, from [140]. The normal system function is carried out by the top four gates of latch L1, under the control of the system data input D and the system clocking control C, producing the normal output +L1. The overhead for implementing scan design in this SRL includes the whole of latch L2, the

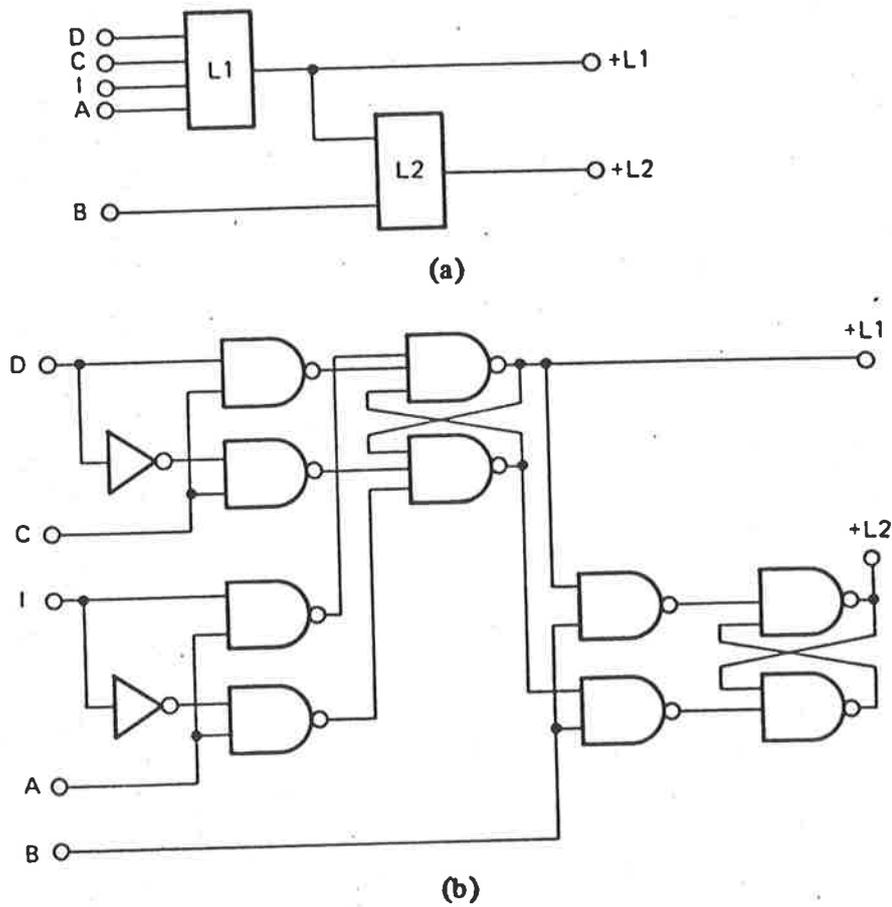


Figure 2.6: Symbolic and Logic Diagrams of LSSD Shift Register Latch (SRL)[140].

lower two gates of latch L1, the clock lines A and B, and the scan data input I and output +L2.

Considering the overhead of applying this scheme to a complete system, Williams and Parker [140] report that in their experience of LSSD products, the overhead has been in the range 4 to 20%. The spread of overhead values results from differing extents to which the designers utilised existing system latches for the L2 latches. For example, the IBM System 38 literature reports that in this system 85% of the L2 latches were used for system function. This may require a more complex L2 latch than that illustrated here, but this is compensated by the overall overhead reduction.

Nippon Electric Co. (NEC) have developed and implemented their own variant of scan design. Known as Scan Path, it was first reported by Funatsu, Wakatsuki and Arima [54] in 1975. Although it aims to achieve the same result as LSSD, the implementation is somewhat different. The Scan Path utilises a "raceless D-type flip-flop" for its SRLs, as illustrated in Figure 2.7 (from [140]). This differs from the LSSD SRL in that only one shift clock is used, and the slave latch is clocked by the inverse of the system and shift clocks. The dependence on a single clock and its derived inverse to latch the master-slave pair leads to the possibility of a race condition if the output of the slave latch is gated into the input of the preceding master latch. This can be avoided by design constraints on feedback paths, particularly the delay characteristics, and constraints on the skew between the clock and its inverse, and does not appear to be a particularly burdensome problem. NEC have reported the use of the Scan Path scheme, in association with some extra refinements, for a large processor system, the

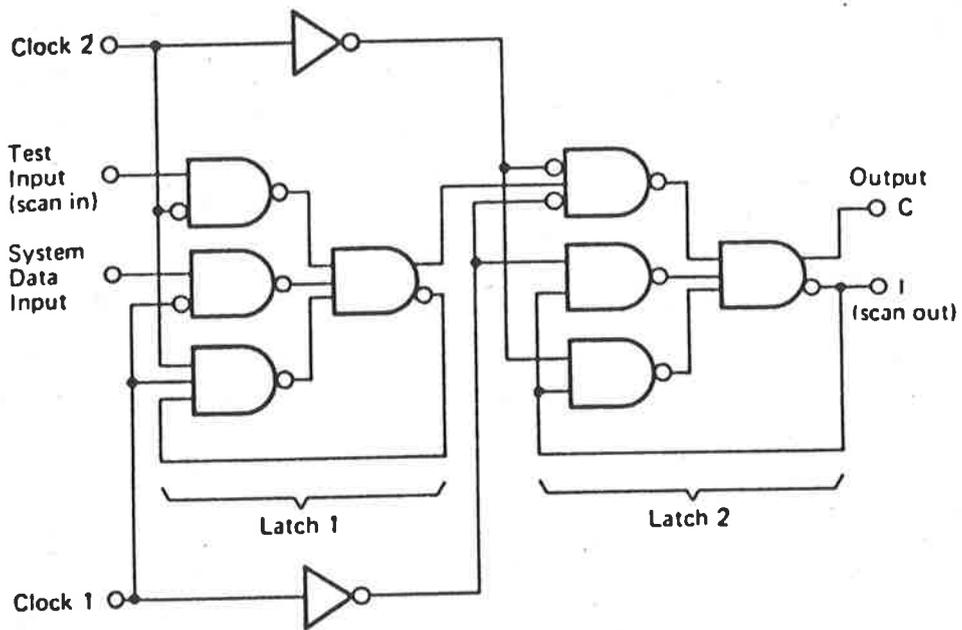


Figure 2.7: Scan Path SRL [140].

FLT-700 System, which contains of the order of 100,000 logic blocks (gates, inverters, *etc.*).

The third variant, Sperry-Univac's Scan/Set logic (also known as bit-serial logic)[127], differs considerably from the previous two in that, while shift registers are used to capture system data and apply inputs at up to 64 points, these registers are completely separate from the system registers. The basic idea is conveyed in Figure 2.8 (from [140]). The system registers of interest are connected to the bit positions of the 64-bit shift register, and data capture is accomplished by a single latch pulse. The shift register may then be scanned out irrespective of the ongoing system function. Furthermore, those system registers connected to the Set data lines of the shift register may be controlled by loading the shift register serially, then pulsing the Set pin to parallel load the system registers.

Note that the system latches are unchanged, except for some additional logic at those latches which must be set from the shift register. The consequence is that the system must still be tested as a sequential circuit, rather than as a set of combinational subcircuits. However the controllable/observable latches impart a measure of testability to the circuit which will ease the task of test generation and fault simulation. The advantage of this scheme over the other scan designs is that the sampling pulse may be applied to the shift register at any time, including during system operation, so a "snapshot" of the sequential machine may be taken and shifted off-chip without any degradation in system performance.

Random-Access Scan [6] has the same aim as LSSD and Scan Path, the complete controllability and observability of all internal latches. However,

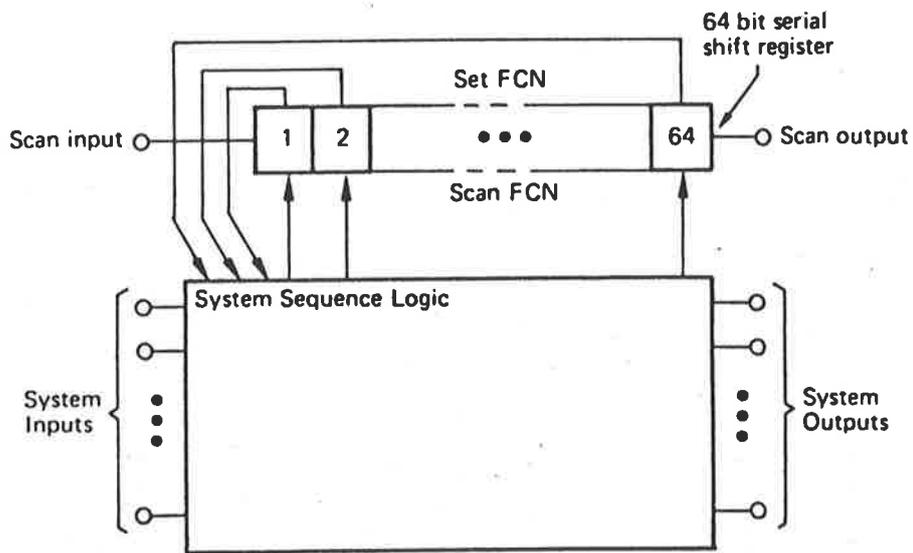


Figure 2.8: Scan/Set (or Bit-Serial) Logic [140].

it is substantially different in implementation. The basic idea is to select one latch at a time, using X-Y addressing, similar to RAM addressing (hence the name). The system latches can be of two types, a dual-port latch or a set/reset latch. Each latch connects to one X-address line and one Y-address line, the distinction between this and a RAM array being that the address lines are routed to suit the latch placement in the circuit, rather than having the very regular cell placement of RAM. The only time a latch responds to the scan clock or to the set/preset signals is if it is selected by having both the X and Y address lines simultaneously high. While the system latches have little overhead to implement this scheme, there is some considerable overhead in the address-line routing, the X and Y address decoders and the primary input/output pins. This latter can be reduced by using a serial scan-in for address selection.

The use of these above techniques, in particular LSSD and Scan Path, aids in the generation of tests for the circuits in two ways. First, the circuit is turned into a combinational circuit by the removal of the latches from the circuit, and second, the ability to control and observe all of the internal latches in effect partitions the circuit into smaller subcircuits. As noted in the previous section, partitioning is an effective tool in reducing the test generation complexity. There exist automatic programs to partition networks which work by starting a partition at the SRLs and back-tracing through the combinational parts of the circuit until either a primary input or another SRL is encountered. In NEC's system, if the back-trace becomes too long, and thus the partition becomes too large, controlled D-type flip-flops are inserted into the circuit, independently of circuit function, in order

to create a partition boundary [140]. It has been reported [142] that NEC have been able to utilise this system to automatically partition circuits, and examples were given of partitioned subcircuits with overlaps in the range 45% to 75%. The circuits partitioned into between 3 and 16 subcircuits, and using a method given in [142], the estimated cost of test generation for the partitioned circuits would be in the range 70% to 20% of the cost for the unpartitioned circuits.

There are three main objections to scan design. Some designers resist the introduction of the methodology because of the increased restraints it places on them in their usage of latches and flip-flops. Despite LSSD and Scan-Path having been implemented successfully in large products (IBM System 38, NEC FLT-700), some authors still feel the need to weaken the disciplines to broaden the acceptance of the methods [60].

The second objection is that the serial shifting in and out of long patterns increases test times, and taxes the storage capabilities of existing VLSI Automatic Test Equipments (ATEs). Both of these can be ameliorated by the use of multiple shorter scan paths, with a minimal loss of overall reliability, and some significant gains in the prototyping ability (see discussion in Appendix C). The problem with the ATE pattern storage is that many existing ATEs are not optimised for the scan type tests, and to hold all inputs constant while scanning in the scan data, the ATEs must apply full width input patterns, one for every bit position in the scan path. This can be overcome by the addition of ATE modules specifically intended for long serial sequences, such as the Fairchild Serial Test Module [44]. Further improvements can be made if the test patterns and expected

responses are stored in a "change-only" format [90]. The test time problem is serious, and is only offset to a minor extent by the fact that the scan design methodologies inherently partition circuits, and greatly reduce the sequential complexity.

The third problem is that as circuit densities increase, so the likelihood of interaction between neighbouring nodes and between adjacent signal lines increases. These transient effects are far more likely to occur during full clock rate operation than during the single step clock operation necessitated by scan testing.

These latter two problems, coupled with the fact that silicon area for test functions is becoming cheaper than off-chip test generation and application, provide the major impetus towards Built-In Self Test.

2.4.3 Built-In Self Test.

Built-in self test can be regarded as another method to partition the circuit, where, instead of applying vectors stored externally and analysing responses externally, circuitry is built in to test the partition with minimal use of external references. The subject of built-in self test encapsulates two main topics: generation and application of test vectors, and capture and analysis of the response. While the concept of incorporating the test structures on-chip is relatively new, the suggested techniques in many cases derive from test methods historically used to diagnose board level systems [85,86,140].

2.4.3.1 Test Generation and Application.

Built-in self test requires the application of test vectors from an on-chip source to the input nodes of a partition under test, with the external control of the test being minimal. The most obvious method is to store test patterns in on-board ROM and externally trigger an on-chip sequencing controller that will bring out the stored patterns and cause them to be applied to the partition. This method has been used in bus-oriented systems, such as microprocessors, generally to perform verification of the main controller, and also to perform a limited verification of the interface between the internal buses and the external pins, thus allowing more extensive external testing to be carried out. A good example of this is the test procedure for the CSIRO "100k" chip, described in [31]. Note that in these cases, the ROM-stored tests are only used to verify small critical areas: once these have been verified, the usual trend is to utilise them to assist in testing the rest of the circuit. To store large tests in on-chip ROM is very space inefficient, and is not generally done. Another problem with ROM-stored tests is that they must be pre-generated, so the worst facet of the external test problem is not alleviated. Also, if at some later stage in production the original test set needs modification, the chip must undergo a mask change to alter the ROM.

The remainder of the test application techniques rely on concurrent generation of the test vectors. The test set types to be considered are exhaustive, pseudorandom and deterministic. An assumption made for the rest of this discussion is that the circuit to be tested may be partitioned into

smaller subcircuits, and those subcircuits are combinational, not sequential. This is not unduly restrictive, because as has been shown in the previous section, the introduction of scan design to the circuit can achieve those conditions. While scan design will aid the implementation of most of the techniques described herein, it is essential to none: they all may be applied in isolation, albeit at a possible cost of increased overhead.

Considering exhaustive testing first, it was noted in Section 2.3.1 that combinational circuits could be tested without test generation by simply applying all possible input combinations. While the test length grows exponentially with the number of inputs, it is generally felt (*e.g.* [85]) that partitions with up to 20 inputs have acceptably small exhaustive test sets ($2^{20} \approx 10^6$ input patterns), provided of course that these are directly applied rather than being shifted in.

As discussed previously, McCluskey and Bozorgui-Nesbat have shown [84,101,102] that certain classes of combinational circuit are partitionable by the sensitisation of certain paths. They also illustrate psuedoexhaustive testing, for cases where a network's outputs are each dependent on less than the complete set of inputs, or a sensitised partition can be applied. The idea is to exhaustively test the output or partition by holding constant the sensitising inputs, and varying all the other inputs. If some inputs have no effect on the partition or output under test, then these can be simultaneously be varied to test other partitions or outputs concurrently, resulting in reduced overall test times. This technique does not require fault simulation, but computations are necessary to determine the test set. McCluskey [84] comments that the time required for this task seems to be

much less than that required for test pattern generation.

The immediately obvious method of applying an exhaustive test to a set of inputs is to utilise a counter of length equal to the number of inputs. This is acceptable if a counter exists in circuit adjacent to the inputs to be tested. It is more usual however to have latches at the inputs to combinational circuits, particularly if the circuit has been rendered combinational by the use of scan design, and these latches can be joined into a shift register. If a feedback loop is formed around the shift register, the shift register can be clocked to produce a recurring sequence of patterns in the register positions. By making the feedback loop in one of the forms illustrated in Figure 2.9, the feedback shift register implements a modulo-2 polynomial division. This linear function leads to the name given to these type of structures, Linear Feedback Shift Registers (LFSRs).

A detailed discussion of LFSRs is given by Bhavsar and Heckelman in [16], and will not be replicated here. Suffice it to say that the feedback loops can be selected to give maximal length sequences: that is, if the shift register consists of n master-slave latch pairs, the maximal length sequence of register contents is a sequence of $2^n - 1$ patterns, each unique in the sequence, and each derived from the previous pattern by shifting the register contents one position. The only pattern from an exhaustive set which is missing from this maximal length sequence is the all-zero pattern. This can be generated by a hardware addition to the LFSR (*e.g.* [83]). Thus an LFSR can usually generate the exhaustive test set with far less overhead than would a counter.

Pseudorandom sequences, as discussed in Section 2.3.3, incur lengthy

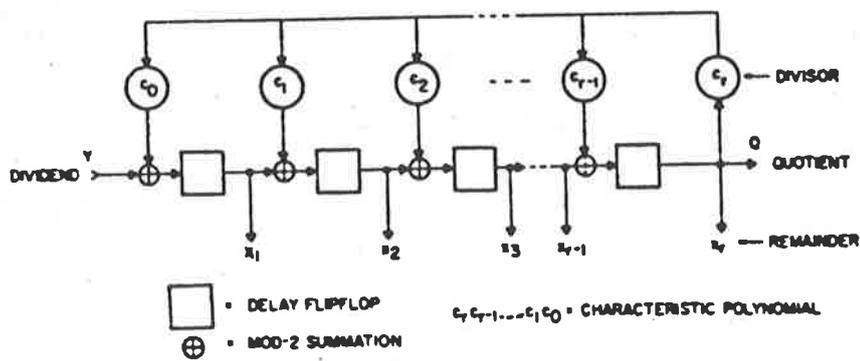
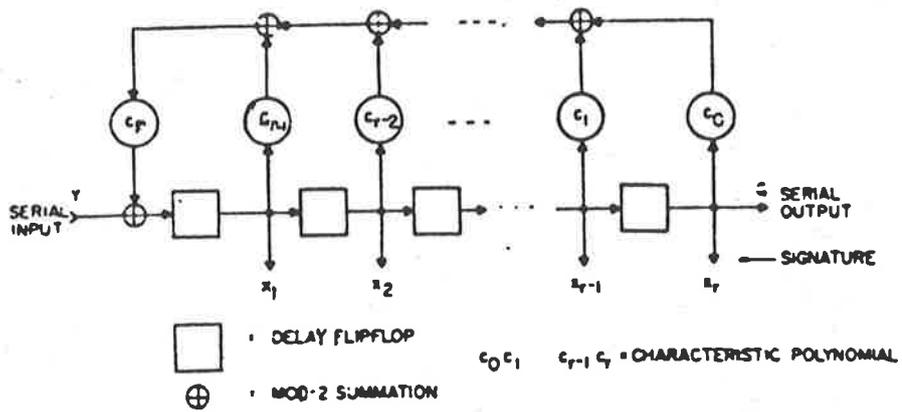


Figure 2.9: Two forms of linear feedback shift register (LFSR)[16].

fault simulations to calculate fault cover and may require long sequences to test particularly difficult faults. Nonetheless, there is still significant industry usage of this test generation method (*e.g.* [69]). The easiest way of generating and applying a pseudorandom sequence on chip is to use a LFSR, as described in [16]. It can be shown that appropriate choice of the feedback polynomial of the LFSR can yield a recurring sequence of less than maximal length, and furthermore the length of such a non-maximal sequence is dependent on the original values in the shift register positions, or the "seed" as it is termed. Hence the length of a pseudorandom sequence may be tailored to suit the circuit's requirement. In general the test would be applied from a LFSR with as many bit positions as the circuit has inputs, and the sequence length would be less than maximal (otherwise no significant advantage would accrue from the use of pseudorandom patterns instead of exhaustive patterns). Illman, of International Computers Limited, described in [69] techniques used in his company's products, and gave an example of a 16 bit carry look-ahead adder tested by 2^{20} pseudorandom patterns, rather than 2^{33} patterns required for exhaustive test, with a single stuck-at fault coverage in excess of 99.9%. In the case where the number of inputs is small, even a maximal length sequence may not be long enough, and extra stages of the shift register, not connected to any input, may be necessary to increase the test length.

Another use for shift registers is for the application of deterministic patterns, either hand or machine generated. Illman [69] described heuristic patterns generated by LFSRs to test certain logic structures which are resistant to random test patterns. Mucha and Daehn [93] described the use

of NON-linear feedback shift registers to generate patterns on-chip, and presented an algorithm that for a given test set, calculates the required register length, feedback function and start vector.

Note that in these methods using FSRs to concurrently generate the test sets, the starting seed is important. This may be initialised by local hardware, such as a hard-wired preset or a small local ROM, or the FSR may be connected into a scan structure, allowing the seed to be scanned in from an external store. This latter method has the advantage of retaining flexibility, so that if a post-design change to the test set is required, a different or additional seed may be scanned in, or in the worst case, extra test vectors may be scanned in and directly applied.

2.4.3.2 Data Compression Techniques.

For the purposes of built-in self test, we need the ability to assess the response of circuit partitions to test sets, while the outputs of those partitions may not be observable from any of the primary outputs. The practicality of altering or enhancing the circuit to directly provide such observability is often dubious, and the observability afforded simply by scan design is often tied to a huge increase in test run times, due to the scanning of long SRLs. Thus we need to either compress the results to reduce the problems of transferring them off-chip, or assess the results on-chip.

Addressing the latter first, although it is in theory possible to generate the expected response concurrently, the only circuits for which this may be readily implemented are those involving modular redundancy or concurrent checking (see Section 2.5.1), or those whose functionality requires numbers

of identical subcircuits (*e.g.* bit-sliced architectures). In the usual case, expected responses are pre-calculated for a given circuit partition and test set. Unless the expected response set is unusually small or regular, the storage of the expected responses in an uncompressed form in ROM is prohibitively expensive in area, and so either the expected responses are stored in a compressed form and expanded for comparison with the test results at test time, or the actual test results are subjected to the same compression as the expected response and are compared in compressed form. The former scheme fails in that the response will vary from circuit to circuit, and so the compression and expansion cannot rely on any general form, and must be optimised for each response set, with the consequent difficulty that the expander hardware must be redesigned for each test set. Thus the only viable scheme is for the compression of the test results, whether the compressed result is assessed on or off chip. Note that “compression” is used loosely here, not in the communications theory sense that no data is lost by compression. All known processes for the reduction of test responses incur some loss of information, a feature of some concern particularly with regards to the assessment of the effective fault cover, and so it is preferable to refer to these processes as *compaction* rather than compression [85].

Many compaction schemes have been studied, some being proposed initially for the diagnosis board level faults but now finding applicability as on-chip response compaction. The techniques may be loosely classed as count functions, spectral techniques, parity techniques and signature analysis (or LFSR) techniques.

The count functions are attractive due to their simplicity of implemen-

tation, and one popular function, transition counting [62], was widely implemented in portable testers. Some of the functions which have been considered [50] are:

$$\begin{aligned}
 c_1(R) &= \sum_i r_i \\
 c_2(R) &= \sum_i r_{i-1} \oplus r_i \\
 c_3(R) &= \sum_i \overline{r_{i-1}} \oplus r_i \\
 c_4(R) &= \sum_i \overline{r_{i-1}} \bullet r_i \\
 c_5(R) &= \sum_i r_{i-1} \bullet \overline{r_i}
 \end{aligned}$$

These functions have not received widespread attention for on-chip compaction, primarily because other techniques seem to offer better performance and/or lower implementation costs.

Spectral techniques are based on the idea of forming some "spectrum" from the circuit's response to the exhaustive input set, and comparing that spectrum to the spectrum of the fault-free circuit. The spectrum and/or the circuit must be designed so that all faults of interest produce spectra differing from the fault-free spectrum. The simplest method, syndrome testing [10,11,114,115] counts the number of ones realised by the circuit under test in response to an exhaustive test set. The circuit must be designed to be syndrome-testable, or the techniques described in [115] must be employed.

Susskind [131] showed that syndrome counting was a method of evaluating one of the Walsh spectral coefficients of the circuit under test. Com-

binational circuits with n inputs can be uniquely represented by a set of 2^n Walsh coefficients, and Susskind proposed the examination of two of these coefficients, one of which was used for syndrome counting, to test for all single stuck-at faults. Here too the circuit must be of a form that will return a different spectrum to the fault-free version, if a fault is present.

Muzio and Miller [97] proposed that, rather than redesigning a circuit to make it testable by syndromes or Walsh coefficients, the spectral coefficients to be examined should be selected to give the required fault cover: the spectrum would consist of any suitable combination of the 2^n coefficients.

The prime objection to the spectral techniques of data compaction is their requirement of the exhaustive test set as input. As discussed previously, solutions to the problem of partitioning circuits are not well defined, and hence it is not always possible to get an adequately small partition to contain the test run times within reasonable limits.

Parity techniques are demonstrably inadequate on a stand-alone basis. However, their use has been discussed in conjunction with other compaction techniques. Benowitz *et al* [15] compared parity, signature analysis and combined parity/signature analysis techniques under pseudorandom test sets, and concluded that no advantages accrued by the use of parity techniques. Conversely, Carter [23,24] examined the use of a parity technique in conjunction with syndrome summation and signature analysis techniques, under an exhaustive test set, and demonstrated high values for stuck-at fault detection. Note that once again this technique is limited by the applicability of exhaustive tests.

The most common implementation of data compression for built-in self

test is the signature analysis or LFSR technique. For a single output circuit, this comprises of feeding the circuit output into the input of a Linear Feedback Shift Register (LFSR) such as either of the two LFSRs illustrated in Figure 2.9. The term *signature analysis* was coined by Hewlett Packard to describe the technique's use in their product, the 5004A Signature Analyser [47,98], although Benowitz *et al.* had previously referred to it as *cyclic code checking* [15].

The usefulness of signature analysis stems from the fact that the final state of the LFSR registers is dependent on the input bit stream, and thus if the circuit's output bit stream contains an error, the final LFSR state, or *signature*, is likely to be different to that for the fault-free circuit. However, it is possible for an error to occur in the circuit output, and yet still have the same signature as the fault-free circuit. This is referred to as *aliasing*, and the output sequences that can cause this aliasing depend on the structure of the LFSR: Benowitz *et al.* give a more complete discussion [15].

For an adequately long random input bit stream, the probability of an error being undetected due to aliasing is $\approx 2^{-r}$, where r is the number of register in the LFSR. The problem with this analysis is that the output of a circuit cannot be guaranteed to be random, particularly in the presence of a circuit fault. Although experience with the Hewlett-Packard product and simulation studies of designs for built-in self test [105] have not revealed any applications in which aliasing is a problem, it has neither been possible to derive any general properties of the effective fault cover obtained by signature analysis, nor to derive any simple relationships between circuit faults and resultant errors in the bit streams. A discussion of these problems

is given by Smith [120], with attention being paid to the treatment of more likely non-random faulty output sequences such as burst errors or regularly recurring errors.

The foregoing has referred to signature analysis for single output circuits, or *serial* signature analysis. In many cases the circuit or partition under test has multiple outputs, and it would be impractical in terms of hardware overhead to test each output with separate signature analysers. Benowitz *et al.* suggested two methods to overcome this: one, to multiplex a single LFSR input to all the M outputs and repeat the test patterns M times, and the other, to use a multiple input LFSR, as illustrated in Figure 2.10. The latter technique has been widely accepted, and is usually known as a *parallel* signature analyser, or a multiple input signature register (MISR).

The problems of aliasing are even more pressing for MISRs, due to an extra source of alias error: the combination of an error in output Z_i at time t_j , followed by an error in output Z_{i+h} at time t_{j+h} , has no effect on the final signature [125,61]. Fault coverage studies for the MISR have also been reported on by Könemann *et al.* [75].

A point in favour of signature analysis is that it does not rely on having any particular input test set to stimulate the circuit, and provided that the fault-free response can be calculated, any test set giving adequate fault cover may be used. In particular, it is suitable for the compaction of the results of both deterministic and pseudorandom test sets.

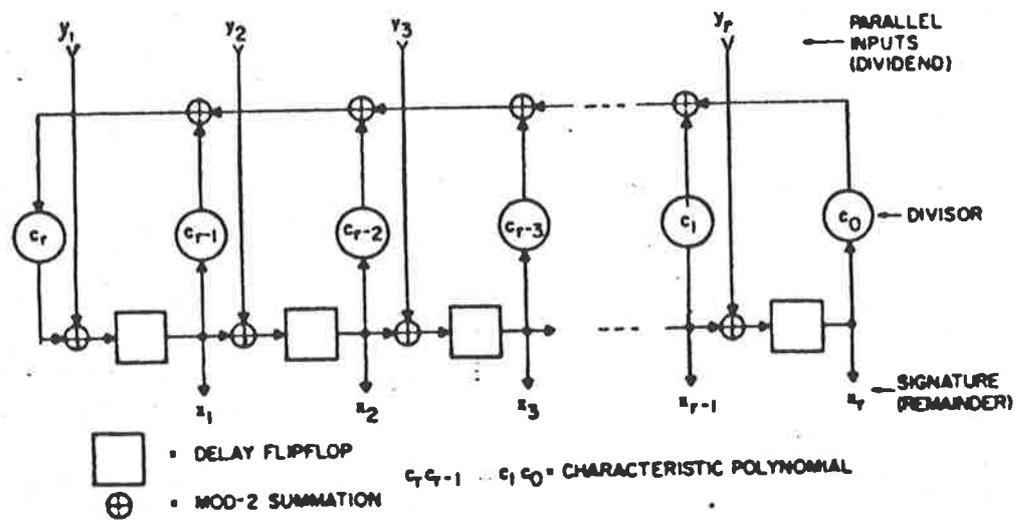


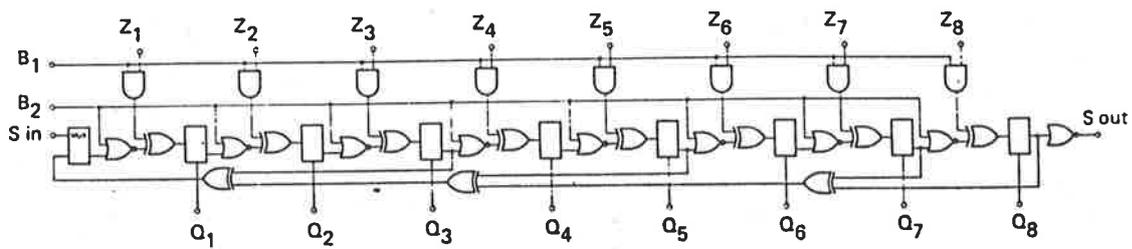
Figure 2.10: Multiple input linear feedback shift register (MISR)[16].

2.4.4 The Built-In Logic Block Observer.

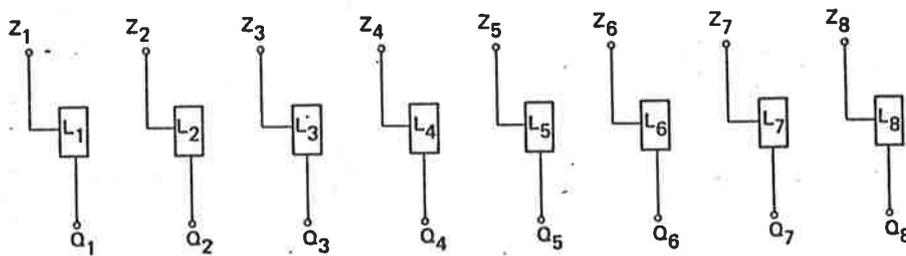
The most widely used technique for the implementation of built-in self test is the application of pseudorandom test sets and the use of signature analysis for compaction. A general testing structure was proposed by Könemann, Mucha and Zwiehoff [74] to implement these functions with little extra overhead beyond that required to implement scan design into the functional circuit. The resultant circuit, the Built-In Logic Block Observer or BILBO, is illustrated in Figure 2.11.

By setting control lines A and B to appropriate values, the BILBO can be configured in its normal operational mode, as a group of independent latches on the boundary between two partitions, or as a scan path section, simply feeding data from latch to latch, or feedback can be added to turn the shift register into a LFSR for pseudorandom sequence generation, or the parallel inputs to the LFSR can be enabled to form a MISR for data compression. While in some cases the implementation of this general structure may lead to functional redundancies, if the BILBO is placed at a totally embedded partition boundary, then it can act as the compaction circuit for the preceding partition, the test generator for the following partition, and the scan path can be used to shift out the compacted signature and shift in the seeds for the pseudorandom test sequences.

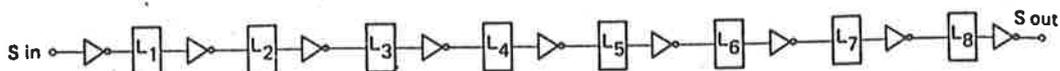
The BILBO concept assumes that the circuit is partitionable into suitably sized subcircuits to allow pseudorandom testing. An extension of the BILBO concept to board and system level has been proposed by Fasang [42,43]. Daehn and Mucha [33] have proposed an extension of BILBO tech-



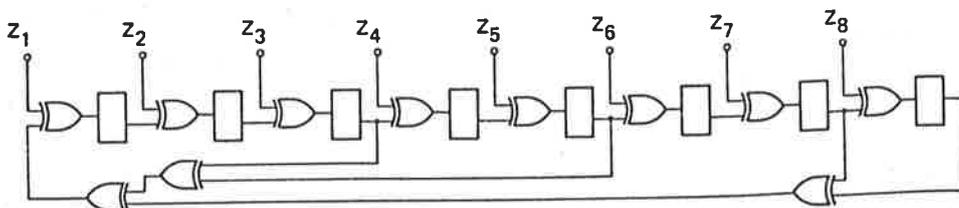
(a) General form of BILBO register.



(b) $B_1B_2 = 11$ System orientation mode



(c) $B_1B_2 = 00$ Linear shift register mode



(d) $B_1B_2 = 10$ Signature analysis register with multiple inputs (Z_1, Z_2, \dots, Z_8)

Figure 2.11: Built-In Logic Block Observer (BILBO) [140].

niques for the testing of PLAs, although more efficient schemes based on universal test sets are now being researched [132].

2.4.5 Built-In Self Test Architectures.

As well as the BILBO-based architectures [74,75,91,42] and other test architectures [101,114,118] based on partitioning, there have been several proposals [107,76] incorporating LFSR test generators and signature analysers, but differing in that they propose that the test hardware resides solely outside the functional region of circuitry, at the package pins. The idea is to implement an external or boundary scanpath which consists of scan path elements connected to the bonding pads. These can be reconfigured into LFSRs and MISRs attached to the primary inputs and outputs respectively, and can self test the circuit. The boundary scan itself is useful for performing interconnect checks between chips during a system test, and may be used by wafer probers to sort good chips prior to dicing in a wafer-level test with a minimum of probe contacts [146]. A recent commercial semicustom part, the National Semiconductor SCX6260 CMOS gate array, incorporates this technique [107], and it is the customer's choice whether it is used or not.

Another approach to chip, module and system testability was proposed by Tsui in [133]. This is based around the use of scan paths, with provisions for single and multiple cycle test mode run controls, initial conditions control and test result retention, with multiplexors being used to overcome any latch shortfall, and is essentially a systems level extension of scan de-

sign. The aim of the approach is to provide sufficient controllability and observability to ensure that the chip, module and system are all testable *in situ*: that is, while all parts are connected, and without the use of much, if any, external test equipment. No restriction is made on the test type to be used in this scheme.

2.5 Other Test Methodologies.

2.5.1 Concurrent Checking.

In the foregoing discussions, it has been assumed that testing is carried out off-line (*i.e.* not during the chip's normal operation), and that the test circuits are to a large extent independent of the circuit's normal function. A methodology that takes a different approach is *concurrent* testing: the test function occurs in parallel with the normal operation of the circuit.

The most common example of concurrent checking is the use of coding to detect and correct errors in data [22, chapter 5]. However, most commonly used codes, such as Hamming codes, Reed-Solomon codes and cyclic redundancy codes, are not preserved by arithmetic operations. Another class of codes, residue codes, have the property that for addition, subtraction and multiplication, the check bits of the result can be determined from the check bits of the operands. Hence, by incorporating a circuit which operates on the check bits in parallel to the arithmetic operation on the data, concurrent checking of the arithmetic circuit may be achieved.

To detect errors in the codes, a checking circuit must be used, and it

is desirable to implement this in such a way that if a fault occurs within the checker during operation, an indication is given externally. This is the principle that is embodied in the design of Totally Self-Checking (TSC) checkers. Such a circuit must be

1. Fault Secure: for a prescribed set of faults in the checker, the occurrence of any code input to the checker will cause either the correct code output or a non-code output, and
2. Self-Checking: for a prescribed set of faults in the checker, each fault has at least one code input that will produce a non-code output if the fault is present in the checker.

The usual assumption of single faults only discounts the case where a non-code input is applied to a faulted checker. The properties of the TSC checker are such that in the presence of a fault in the checker, it will keep on working as if it is unfaulted until such time as the code input corresponding that fault appears at its inputs, at which stage it will indicate a fault. This behaviour is highly desirable in fault-tolerant computing and high-reliability applications, and consequently much work has been done in this area [121,38,100,48,106,68]. Concurrent error detection in general VLSI systems is also subject to much research [73,26,128,32].

In certain critical applications, such as aerospace avionics, an electronics failure could have catastrophic implications. To maintain a high level of reliability, the circuits must be constantly monitored for failure. A traditional approach to this is to employ redundant circuitry, so that all circuit functions are carried out in parallel by three (or more) devices. This is

known as Triple (or in general, N -) Modular Redundancy. This facilitates the on-line monitoring process, since all N circuit outputs can be compared, using a majority voting scheme. Usually, any machine whose outputs differ from the majority is discarded, until the system has too few redundant units to carry in in that mode. The off-line testing is aided too, in that, provided the design has been verified previously, the modularly redundant units can be checked by comparing their outputs to each other. The use of TMR within a single VLSI chip may be of dubious advantage, however, due to the sources of common mode errors, such as powerline spikes and clock failures.

2.5.2 Other Possible Test Techniques.

Some efforts have been made to find design methods which ensure that easily testable circuits are produced. One approach is to constrain the design to a particular class of structures. An example of this is given by Abraham and Gajski [2], in which cellular tree structures are used to automatically implement designs specified in a high level description language. These types of schemes rarely seem to get implemented in real systems because of the excessive constraints they place on logic structures.

Another idea that has been mooted for CMOS design is the utilisation of the CMOS analogue properties of negligible supply current and wide operating voltage range. Levi [77] proposed a scheme which relies on the examination of the supply current while inputs are being applied to the circuit to sensitise nodes. He details tests for non-equivalent shorts, excessive

leakage, opens and shorts between logically equivalent nodes. The tests do not indicate correct logical operation in any way, but rely on the fact that the design has been previously verified, and hence the circuit under test will conform to that design unless a fault, which is detectable by its analogue effect on the supply current, is present. While the concept shows promise, the majority of test strategies currently in use are geared to assessing the digital response, and the acceptance of a radical departure from this is unlikely, in the short term.

2.6 Summary.

An overview of test techniques has been given, showing that the trend in coping with the testing problems of ever-increasing circuit size, complexity and speed is to attempt to partition circuits into smaller subcircuits that are manageable, in terms of test generation and test run times. The techniques used to partition the circuits are reviewed, and it is noted that built-in self test methods are gaining a wider acceptance as a consequence of the large generation times of ATPG testing, and the large test run times of scan design testing, becoming too costly for the VLSI manufacturer.

Chapter 3

The Transform and Filter Brick.

In this chapter, the Transform and Filter Brick (TFB) chip is described to provide a basis for the discussion of the testability issues in the following chapter.

The design has been undertaken as a group project by postgraduate and undergraduate students in the Department of Electrical and Electronic Engineering at Adelaide University, under the supervision of Dr. Kamran Eshraghian. Various members of the design team were responsible for different aspects of the project. Broadly speaking, the logical architecture was specified by David Fensom, Rod Bryant and Bill Cowley, the physical architecture was specified by Alex Dickinson and the author, the multiplier/divider and the adder/subtractor/shifter/accumulator were designed by Greg Zyner, the control store decoders were designed by Michael Pope,

and the execution controller was designed by Paul Franzon. The input and output processors were designed by Greg Zyner and Eddy Savio, the RAM arrays by Michael Pope and Michael Obst, and the data memories by Neil Murray, Greg Zyner and Michael Obst. Paul Franzon was responsible for the work on yield analysis and redundancy. The testability issues have been solely the responsibility of the author.

The issues covered in this chapter have been previously reported in a number of papers and reports. Two papers covering the complete chip [40,41], and a paper containing a discussion of the TFB multiplier/divider [150] have been published. Internal technical reports have been written on the following subjects:

- the logical architecture specification [119],
- the physical architecture specification [35],
- the TFB ALU implementation [147], which covers
 - the TFB multiplier/divider and
 - the TFB adder/subtractor/shifter/accumulator,
- the TFB input and output processors [113,148],
- yield analysis and redundancy considerations [46],
- the RAM arrays and their structure generator program [103], and
- the data memories and their associated control circuits [149,96].

This chapter draws heavily on these sources, and their use is gratefully acknowledged.

3.1 Design Overview.

The Transform and Filter Brick (TFB) was designed to perform a target set of real-time signal processing tasks, such as correlation, convolution, both adaptive and non-adaptive digital filtering, and Fast Fourier Transforms. As a general purpose device, TFB required the capability to operate efficiently both as an element in highly parallel multiprocessor systems, such as are used to process high data volume, high throughput tasks, and as a stand-alone processor for less demanding applications.

3.1.1 Benchmark Tasks.

From the set of target applications, two tasks were chosen as benchmarks, against which the processor architecture was optimised. The first, the Sum Of Products computation, involves purely real arithmetic, while the second, the Fast Fourier Transform, requires the manipulation of complex numbers. It was envisaged that both processes could be realised as Skewed Single Instruction Multiple Data (SSIMD) multiprocessor systems, such systems having been shown to be the optimal multiprocessor solution for many classes of signal processing tasks [8]. In such systems each constituent processor runs the same program, but in skewed synchronism with other processors. The proportioning of the architecture was heavily influenced by the acceptance of this approach to multiprocessing, and the concept of autonomous I/O processors was designed to facilitate the creation of SSIMD multiprocessor systems.

3.1.1.1 The Sum Of Products Computation.

The Sum Of Products (SOP) computation is integral to correlation, convolution and transversal filtering, and so a processor to perform the SOP efficiently could have wide applicability. The SOP computation can be illustrated by considering the correlation process, which requires the evaluation of

$$R_i = \sum_{n=1}^N s_{n-i} r_n \quad i = 0, 1, \dots, M, \quad (3.1)$$

where R_i , the correlation at lag i , is calculated as the sum over N samples of the product of s_{n-i} , the sampled signal delayed by i samples, and r_n , a reference vector. Each step in this task requires the fetching of a signal sample and the corresponding reference weight, and the performance of a multiply-accumulate operation on those two values.

3.1.1.2 Fast Fourier Transform.

The Fast implementation of the discrete Fourier Transform (the FFT) requires the evaluation of a sequence of complex-valued "butterflies" of the form

$$a' = a + bw \quad \text{and} \quad b' = a - bw. \quad (3.2)$$

The multiplication and accumulation to evaluate a' is a SOP step, as is the evaluation of b' . Expanding the complex variables into their constituent real and imaginary parts, the complex SOP can be written as

$$r + js = (r_1 + js_1)(r_2 + js_2) + (r_3 + js_3) \quad (3.3)$$

$$= (r_1 r_2 - s_1 s_2 + r_3) + j(r_1 s_2 + r_2 s_1 + s_3). \quad (3.4)$$

To perform this computation quickly, parallel evaluation of the products and the sums and differences may be carried out, using four independent multipliers and two adder/subtractors. The parallelism may be further enhanced if the values r_1, r_2, s_1 and s_2 are supplied from four independent memories via four independent buses.

Other demands on the architecture are imposed by the difference in value of the weight (w in Equation 3.2) in successive butterflies, and by the large inter-chip data bandwidths required in an array implementation of the FFT.

3.1.2 The Logical Architecture.

A logical architecture capable of performing the target tasks was proposed, and is illustrated here in Figure 3.1. The architecture is separated into processing and control sections. For clarity, the control section, envisaged as a program store, decoder and program controller, is not shown in the figure. Apart from control signals to drive the processing elements, and status flags from the elements back to the control, the only connection between control and processing sections is provided to down-load data from the program store to the processing elements (such as the values of fixed constants or weights).

As noted in the previous subsection, a complex SOP operation may be implemented with a high degree of parallelism if four multipliers, four data memories, four independent data buses and two adder/subtractors are used. The manner in which the buses are connected to the processing elements

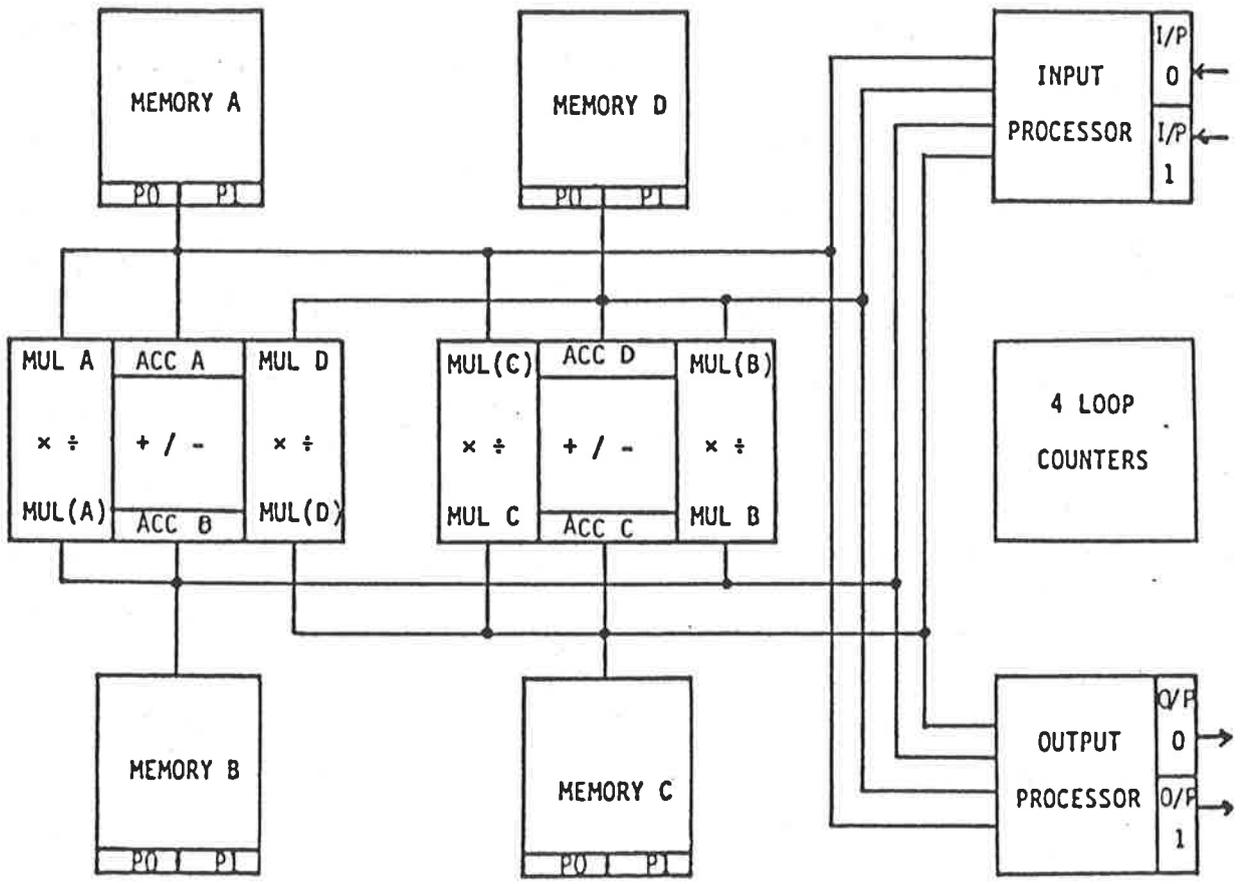


Figure 3.1: TFB Logical Architecture.

in Figure 3.1 facilitates the parallel loading of all operands for a complex SOP in one instruction cycle. Not explicitly shown in the figure are extra switching capabilities required to achieve fully parallel SOP operations: these are the ability to swap the contents of the two accumulators associated with a particular adder/subtractor, and the ability to pass a word to the input or output processor and then read it back onto another bus. Both of these operations are necessary to achieve connectivity between separate data memories and buses.

The processor is further enhanced by making the multipliers reconfigurable, under program control, as dividers. This widens the applicability of the processor by allowing the efficient implementation of certain classes of algorithms, including several important lattice filters.

The separate input and output processors are designed to facilitate array operation, while enhancing the overall flexibility of the chip. Both support a set of I/O modes, both synchronous and asynchronous, and the handshaking associated with these I/O modes is handled completely by the separate processors, relieving the central control section of the chip of tedious "housekeeping" tasks. The I/O modes include the capability of splitting the ports into two half-word ports, with independent operation of the two halves. Another option is for single byte or double byte communication, both in one processor clock cycle, with automatic word splitting and reassembling. Combining double byte communication with the half-port connection scheme allows a full word input from each of two devices, and a full word output to each of two devices, all in one processor clock cycle. This high inter-chip data bandwidth is essential to efficient array

operation.

Another means of removing "housekeeping" tasks from the central processor is employed in the data memories. Two independent pointers address each memory, and each pointer may automatically increment in a number of ways. This allows the programmer to set up regular data structures, and the recursive execution of the program utilises the auto-increment capability to select the current data.

Using the logical architecture suggested here, together with an "instruction set" specifying the operations required of the processing elements, multi-processor arrays were investigated, and various algorithms were coded. These studies resulted in the dimensioning of the data memories and the control program store RAM array.

In the typical multi-processor applications studied, the data bandwidths are high enough to allow the data flow to keep pace with the processing. Thus the data memories need only be large enough to store the immediate processing requirements and those constants which are to be re-used, and 32 words per data memory was found to be an adequate size. The data word size was chosen as 16 bits, allowing for easy interfacing with many current devices, both digital and hybrid, and the two's complement representation was selected to facilitate the arithmetic processing. A program storage of 128 words was found to be sufficient for all algorithms coded, including complex algorithms such as multi-stage adaptive lattice filters. (For example, a four stage gradient type adaptive lattice filter required only 64 words of program store.) The width of the program word could not be determined at this stage, however, as it is highly dependent on the

physical implementation of the logical architecture.

3.1.3 Transformation to the Physical Architecture.

The transformation of the proposed logical architecture into a physical architecture was performed with the aims of maximising the circuit's regularity, planarity, and hierarchical partitioning while minimising the intercell routing and enhancing the performance, testability and yield as much as possible [136].

A major obstacle to this mapping arose from the fact that the logical architecture was not a planar graph: that is, there is no way to arrange the various elements and their interconnections without incurring a cross-over in the communication paths. The implication of this in the physical architecture is that, if fully parallel communications are used, one has to arrange the cross-over of at least two 16-bit buses: a procedure wasteful of much needed silicon area. Another point of concern was the large area and power consumption needed to implement four fully parallel array multipliers. These problems can both be solved by combining the adoption of parallel/serial multipliers (in which one operand is supplied in parallel and the other is supplied serially), the use of parallel to serial conversion registers as remote or delocalised input devices for the multipliers' serial inputs, and the grouping of a multiplier, an adder/subtractor/accumulator, a data memory and a delocalised multiplier input (DMI) on a single bus segment. The TFB physical architecture based on this solution is illustrated in Figure 3.2, with the blocks labelled ALU representing a multiplier/divider and

an adder/subtractor/accumulator sharing a common bus.

Key elements in this architecture are:

- the implementation of four separate identical processing structures,
- the ability to either carry out parallel local operations on the disjoint bus segments or to connect several bus segments together to effect global communications,
- the use of the serial paths supplying the multiplier inputs to minimise the cross-over problem (*i.e.* maximise the planarity) while maintaining the connectivity required for complex SOP operations, and
- the autonomous nature of the multipliers' operation.

This system of parallel arithmetic computation on-chip represents a substantial departure from the norm for commercial signal processing chips. Commercial processors utilise single very fast multipliers, which return a full precision result in one processor instruction cycle. These occupy large silicon areas and are only active for a small portion of the run time (20-25% for typical signal processing algorithms). The latter fact means that the multipliers' apparent speed is never realised in an algorithm.

The multipliers in TFB are relatively slow, producing a 32 bit result from two 16 bit operands in eight clock cycles, with a further cycle being required for latching the result. However, the multipliers, once started, carry on autonomously until their result is produced. This frees the system buses, the I/O and the control for other tasks while the multiplications are being carried out. In most algorithms the multipliers can be running

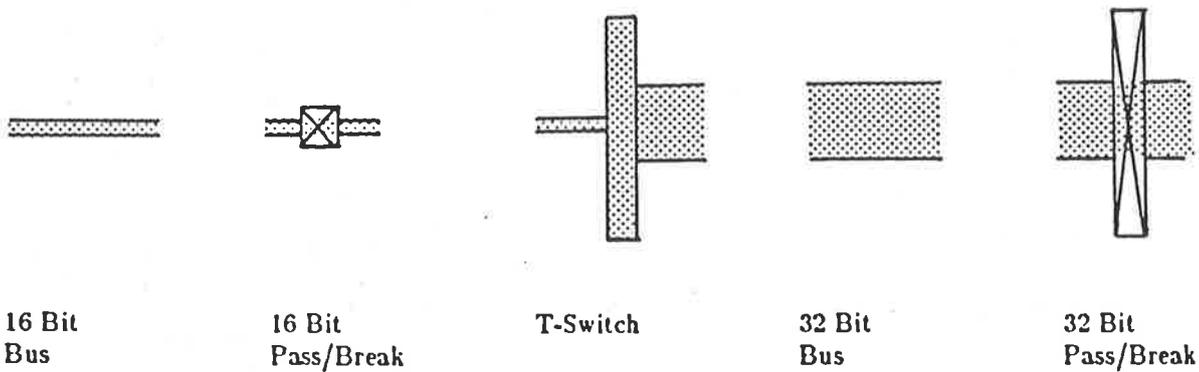
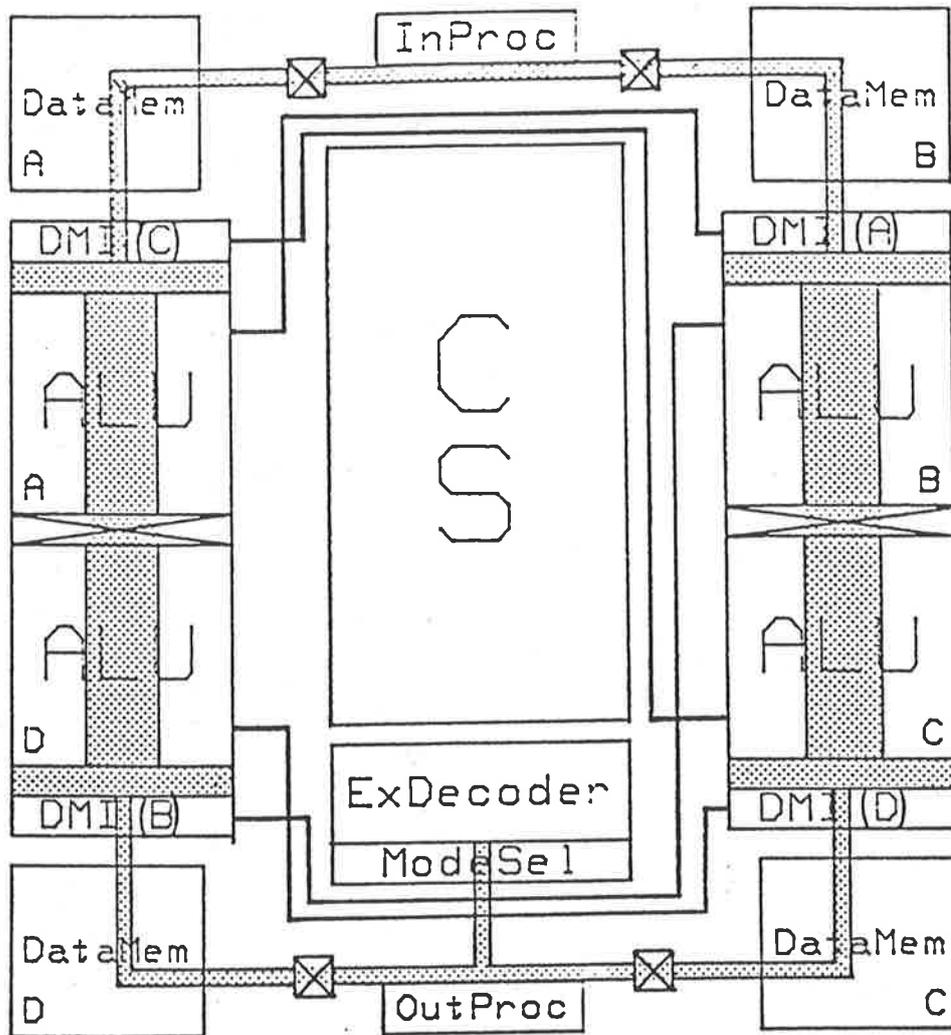


Figure 3.2: TFB Physical Architecture.

continuously because all other arithmetic and data transfer tasks can be pipelined to occur during the multiplication time.

This pipelined parallel operation brings the effective time per chip for a single multiply-accumulate operation down to $280ns$ (at the projected clock speed of $8MHz$). This compares favourably, for instance, with the $400ns$ required by the Texas Instruments' TMS32010 to execute a multiply-accumulate in straight line coding, particularly as extra time penalties are paid by the TMS32010 to perform looping, data manipulation, I/O or further calculations typically required between successive multiply-accumulate operations in a recursive algorithm.

The clocking strategy is based on a three phase non-overlapping clock, primarily to ease the communication problems associated with the highly interconnected parallel processing elements. Phase one is reserved for passing information around the Ring Bus, and for evaluation of any conditional expressions that may exist at that program step. Phases two and three are primarily operation cycles.

Having found a transformation to a physical architecture which conserves or enhances all the desired features of the logical architecture, the physical architecture was then specified more completely [35], to allow the designers to work on subsections of the design within known functional, performance and interfacing constraints. The next two sections discuss the details of the TFB design. Due to the segmentation of the design into control and data handling sections, these will be treated separately.

3.2 The Data Handling Elements.

3.2.1 The Ring Bus.

Central to this architecture is the Ring Bus which allows parallel communication between any and all of the processing, data storage and I/O elements on chip. As shown in Figure 3.2, it consists of a 16 bit bus connecting the Data Memories and Delocalised Multiplier Inputs to the I/O processors at each end, multiplexed to a 32 bit bus embedded in the Arithmetic Logic Units.

The bus may be switched into discrete segments by the Pass/Break switch arrays and the 16/32 bit T-switches shown in the figure. The T-switches incorporate a sign-extend/zero-fill capability, and have the following possible states:

Break: isolates the 16 bit bus from the 32 bit bus, and does not force any values onto any bus segments,

Pass Upper: bidirectionally connects the 16 bit bus to the upper byte of the 32 bit bus, and places a 16 bit zero fill on the lower byte of the 32 bit bus, and

Pass Lower: bidirectionally connects the 16 bit bus to the lower byte of the 32 bit bus, and places a 16 bit sign extend in the upper byte of the 32 bit bus.

The state of each T-switch is controlled by pairs of instruction microcode bits: *Pass/Break* and *Upper/Lower*. The *Pass/Break* switches are each

directly controlled by single bits in the instruction microcode (one bit per switch). The programmer directly selects which bus segments to connect together to perform the required communication task for each instruction. While this adds to the programming overhead, it maintains a flexibility which would be lost by using higher level path selection instructions.

3.2.2 The Data Memories and Pointers.

Each of the four Data Memories stores 32 words of 16 bits, and has two pointers that independently select any address in memory. Every pointer has an associated data buffer which is independently accessible from the bus. Data is read from the bus into one of the buffers and then into memory, or vice versa. A Data Memory can perform Read/Write operations with automatic pointer updating, loading of the pointers' control circuits with start addresses and preset increments, and null operations. The sixteen possible operations, detailed in Table 3.1, are selected by a fully encoded four bit instruction microcode field. There are four such fields, one for each Data Memory.

All memory operations fit in with the global communication scheme, in which data is read to the bus on phase one, and latched from the bus on phase two. As the Read instruction is not available to the Data Memory until the beginning of phase one, the memory uses a lookahead scheme to ensure that valid data is held in the data buffers at that time, and during phases two and three respectively the selected pointer and its data buffer are updated. For a Write operation, the pointer update and data buffer

| Mnemonic | Function |
|----------|---|
| NULL | Nothing |
| RP0I0 | Read with pointer 0 and do not increment address |
| RP1I0 | Read with pointer 1 and do not increment address |
| RP0I1 | Read with pointer 0 and increment address by +1 |
| RP1I1 | Read with pointer 1 and increment address by +1 |
| RP0IP | Read with pointer 0 and increment address by preset |
| RP1IP | Read with pointer 1 and increment address by preset |
| WP0I0 | Write with pointer 0 and do not increment address |
| WP1I0 | Write with pointer 1 and do not increment address |
| WP0I1 | Write with pointer 0 and increment address by +1 |
| WP1I1 | Write with pointer 1 and increment address by +1 |
| WP0IP | Write with pointer 0 and increment address by preset |
| WP1IP | Write with pointer 1 and increment address by preset |
| LIP0C | Load Immediate pointer 0 control word |
| LIP1C | Load Immediate pointer 1 control word |
| LIPBC | Load Immediate both pointer 0 and pointer 1 control words |

Table 3.1: Data Memory Operations.

write occur on phase two, with the memory word being written from the data buffer on phase three.

The pointers and their associated data buffers are updated each time they are selected for a Read or a Write. The pointer update is specified by the instruction, and can be an increment of zero, one or a preset increment in the range -16 to +15. Each pointer has independent control circuitry, which can be loaded with a unique start address and preset increment, using data placed on the bus by the instruction microcode. This pointer control data consists of a five bit address field, an address field enable bit, a five bit preset field, and a preset field enable bit, allowing any memory to renew either or both of its start address and preset increment, or neither. If the pointer's start address is loaded, the associated data buffer is updated to keep the lookahead scheme intact.

The basic memory cell used is a generalisation of the simple 6 transistor static CMOS cell, incorporating a double word line. This allows the creation of a "doubly decoded" array, in which both memory pointers decode in parallel, using separate decoders, with only one being finally selected to address memory. The data memory floorplan is illustrated in Figure 3.3.

The pointer control circuit, assembled from 5 bit-slices, consists of a full adder, a current address register, a previous address register, a preset increment register, and logic to allow the loading of those registers and the selection of inputs to the adder. The adder has as inputs the previous address register, and a choice of either zero, one (a least significant bit carry in) or the preset increment register. The current address register may be loaded with the adder result, or the current address may be loaded from

the bus. The current address is passed to the address decoder.

3.2.3 The Arithmetic and Logic Units.

The blocks in Figure 3.2 labelled ALU, the Arithmetic and Logic Units, each comprise of a multiplier/divider and an adder/subtractor/shifter/accumulator (ASSA). Both of these will be discussed in detail in the following subsections. It should be noted here that not all possible combinations of actions of the multiplier/divider components and ASSA components are considered necessary or desirable, and hence only a restricted set of thirty four "ALU instructions" is allowed. These are coded as five bit fields in the instruction microcode, with a separate field for each ALU. Table 3.2 details the instructions.

It can be seen from the table that some of the operations are "double ALU" instructions, in that they require operations in two adjacent ALUs, separated by a 32 bit Pass/Break switch. It is the programmer's responsibility to set the Pass/Break and T-switches to appropriate conditions and to select complementary instructions in the two ALUs to ensure the correct operation of the "double ALU" instructions.

Not all of the thirty four mnemonics of Table 3.2 represent unique states of the control lines. Some states have been duplicated because they are required for different types of operation, *e.g.* BRDACC, ADDAC2 and SUBAC2 all have the effect of placing the accumulator contents onto the bus, but the associated operations in the adjacent hardware are different in each case.

| Mnemonic | Function |
|----------|--|
| NULL | No operation commenced in the ALU in this clock cycle |
| HRDACC | Read higher 16 bits of accumulator onto bus |
| LRDACC | Read lower 16 bits of accumulator onto bus |
| BRDACC | Read all 32 bits of accumulator onto bus |
| HWRACC | Write higher 16 bits of accumulator from bus |
| LWRACC | Write lower 16 bits of accumulator from bus |
| BWRACC | Write all 32 bits of accumulator from bus |
| CLRACC | Clear the accumulator |
| SHLACC | Left shift accumulator |
| SHRACC | Right shift accumulator |
| NEGACC | Negate accumulator |
| ADDAX | Add multiplier product to accumulator |
| ADDAX2 | Add multiplier product from adjacent ALU to accumulator |
| ADDA2X | Add multiplier product to the adjacent ALU's accumulator |
| SUBAX | Subtract multiplier product from the accumulator |
| SUBAX2 | Subtract adjacent ALU's multiplier product from accumulator |
| SUBA2X | Subtract multiplier product from adjacent ALU's accumulator |
| MOVAX | Move multiplier product to accumulator |
| ADDACC | Add adjacent accumulator into accumulator |
| ADDAC2 | Add accumulator into adjacent accumulator |
| SUBACC | Subtract adjacent accumulator from accumulator, result in accumulator |
| SUBAC2 | Subtract accumulator from adjacent accumulator, result in adjacent accumulator |
| LOADX | Load multiplier parallel input register with upper byte of ALU bus |
| MOVXA | Load multiplier parallel input register with upper byte of accumulator |
| GOX | Multiplier go |
| LDDDGO | Move 32-bit dividend in from bus, start division. |
| MVDDGO | Move 32-bit dividend in from accumulator, start division. |
| LOADDV | Move divisor from bus into divider |
| NEGBUS | Negate the value on the bus and store in the accumulator |
| INCACC | Increment the accumulator value by +1 |
| LDGOX | Load multiplicand from bus, start multiplication |
| MVGOX | Load multiplicand from accumulator, start multiplication |
| TESMUL | Test version of LDGOX, with single internal addition |
| TESDIV | Test version of MVDDGO, with single internal addition |

Table 3.2: ALU Instruction Set.

The instruction set does not allow direct communication between the external 16 bit bus and the multiplier/divider's Product Register. This restriction is used to avoid situations where bus contention would occur due to the zero fill/sign extend operation of the T-switches.

3.2.4 The Multiplier/Dividers.

As discussed earlier, the implementation of fully parallel multipliers was not possible. On the other hand, the major bottleneck for the signal processing algorithms under consideration is in the multiplication step, so efforts have been made to minimise the run time. To this end, the multipliers for TFB implement a Modified Booth's Algorithm [110], a serial/parallel algorithm which produces a 32 bit product from two 16 bit multiplicands in eight clock cycles (with an extra cycle being required in this implementation for latching the result).

The Modified Booth's Algorithm requires the presentation of one multiplicand in parallel, and two new bits of the serial multiplicand every clock cycle, those bits being the least significant pair of bits not yet input. The parallel input is incorporated in the main body of the multiplier, and connects to the upper byte of the 32 bit bus.

The Delocalised Multiplier Input (DMI) is used to accept a multiplicand in parallel, and during the multiply run time it operates on this multiplicand to convert it into bit pairs as required by the algorithm. The connection between the DMI and the main body of the multiplier is restricted to two control lines to the DMI and the serial bit pair from the DMI.

The multiplicands are loaded under program control from the bus. Once the multiplier is started by the program, the multiplier's own control circuit provides all the control signals required internally to complete the multiplication. Upon completion the multiplication result is placed in the 32 bit Product Register, from whence it can subsequently read onto the bus, and a flag is set in the Processor Status Register.

A non-restoring version of the division algorithm was implemented [63], primarily because of the ease of reconfiguring the multiplier hardware to perform the division. This division algorithm requires $N + 1$ clock cycles to generate an N bit result (*i.e.* in this case a 16 bit result is produced after 17 clock cycles). The divider utilises the Product Register to hold the 32 bit dividend, while the 16 bit divisor is loaded at the parallel input. The result is placed in the Product Register after 18 clock cycles: the extra cycle arises from the latching arrangement at the output. Whereas the multiplier cannot overflow, it is possible for the divider to do so if the 32 bit dividend is too large in comparison to the divisor. In this case the Product Register is written with a sign-correct hard-limited result, while a flag is generated and passed to an external error-flagging facility.

The individual control lines to the multiplier/divider to read the Product Register, write the Product Register and the parallel input, select multiplication or division and start the process, come from the ALU decoder. The DMI parallel load from bus signal comes directly from instruction microcode.

With reference to the multiplier/divider floorplan in Figure 3.4, the heart of the multiplier is the 18 bit adder, composed of six cascaded 3 bit

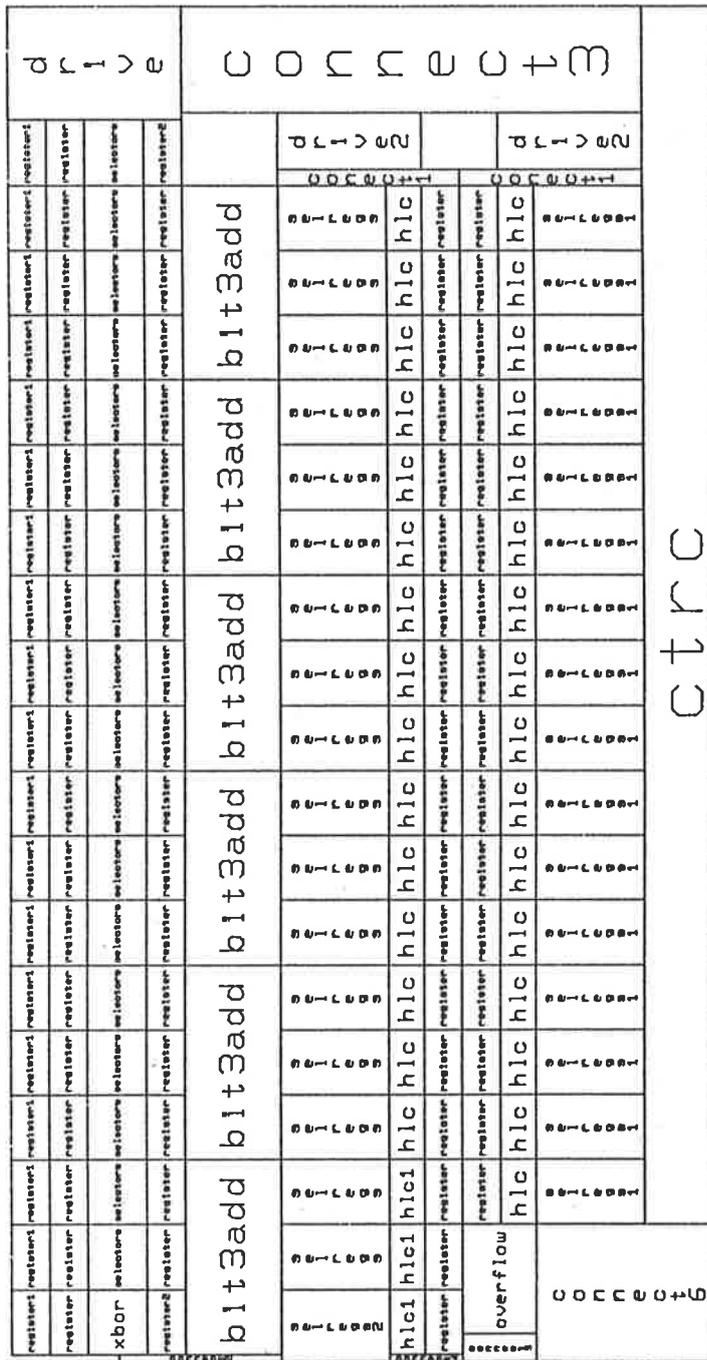


Figure 3.4: The Multiplier/Divider Floorplan.

carry lookahead adder units (labelled "bit3add" in the figure). The Product Register is composed of 34 single bit register cells ("register"), each having an associated hard-limiting cell ("hlc" or "hlc1"). The Parallel Input from the bus is latched into a buffer (composed of "register1" cells), and then shifted into an internal register ("register" cells) at the start of operation. The cells "selectors" perform shifting and negation operations on the parallel input, the function being controlled in accordance with the respective algorithms by the values of the bit pair passed from the DMI for multiplication, and by the output of the cell "xbor" for division. In both cases, the shifted result is held at the adder input in another group of "register" cells. The "selreg" cells act as a partial product register, holding interim results from and passing inputs to the adder, and also as a shift register, for both multiplication and division. Division overflow is detected in the "overflow" cell. The whole system is controlled by the control circuit cell "ctrc", whose main components are some random logic to generate the required control signals, and a shift register used to limit the number of iterations carried out by the system.

3.2.5 The Adder/Subtractor/Shifter/Accumulators.

The Adder/Subtractor/Shifter/Accumulator (ASSA) subsystems each consist of a 32 bit adder, logic stages on the two adder inputs and an accumulator register with single bit shift left/shift right capability. Programmable operations include addition, subtraction, accumulation, negation, incrementation, and left or right shifts, and are detailed in Table 3.2.

The adder, as illustrated in Figure 3.5, comprises eight cascaded 4 bit carry look ahead adder stages. The inputs to the adder may come, via the buffer logic, from the bus, the accumulator register, or both, depending on the instruction selected, but in all cases the result is placed in the accumulator. The input buffer logic stages supply to the adder the number presented at the input to the buffer, or the One's Complement of that number, or zero, whichever is selected by the control circuitry. The One's Complement input is used in conjunction with a least significant bit carry-in to effect a negation or a subtraction. Overflow of the adder is possible, and as in the multiplier/divider it is treated by producing a hard-limited sign-correct result and sending a flag to the external error handling facility.

The accumulator register is a 32 bit register with single bit shift left and shift right capability. It reads to and is written from the 32 bit bus as two 16 bit bytes, either simultaneously or separately. This allows double precision words to be passed into and out of the ALU if required, or the T-switch zero fill/sign extend may be used expand a 16 bit word from outside the ALU to a 32 bit representation in the accumulator. It also reads to and is written from the adder, with the adder input going via one of the input buffer stages.

3.2.6 The Input and Output Processors.

These processors perform data input and output operations under program control, relieving the central control section of the handshaking and sequencing tasks. They are complementary in operation and similar in

| Mode | Synch/Asynch | Width(bits) | 8 Bit Registers Used |
|------|--------------|-------------|--------------------------|
| 0 | A | 16 | 0H and 1L |
| 1 | A | 8 | 0H |
| 2 | A | 8 | 0H then 0L |
| 3 | A | 8 | 1H |
| 4 | A | 8 | 1H then 1L |
| 5 | S | 16 | 0H and 1L |
| 6 | S | 8 x 2 | 0H and 1H then 0L and 1L |
| 7 | S | 8 | 0H then 0L |

Table 3.3: Input and Output Modes.

structure, as illustrated in Figure 3.6. Each processor has four 8 bit registers (0H,0L,1H,1L) grouped in pairs, each pair being connected via switch arrays to one 8 bit port. Data is written to, or read from these registers under the control of the program, but the actual communication with the external devices is handled by the processors. The program initiates input requests to external devices (via the input processor), but the output of data from the TFB chip is initiated by an output request from an external device. The processors only carry out data transfers if valid data is available when the transfer request is made.

There are eight modes of operation, summarised in Table 3.3, and they are selected by a control word loaded from the Ring Bus into a mode control register. The modes, three synchronous and five asynchronous, dictate the form of the data transfer and the connectivity of the registers to the ports. Four of the modes specify that the contents of the register pairs are to be sent through the same port in succession, which effectively transfers a 16 bit word as two bytes through an 8 bit port (or two words through two ports).

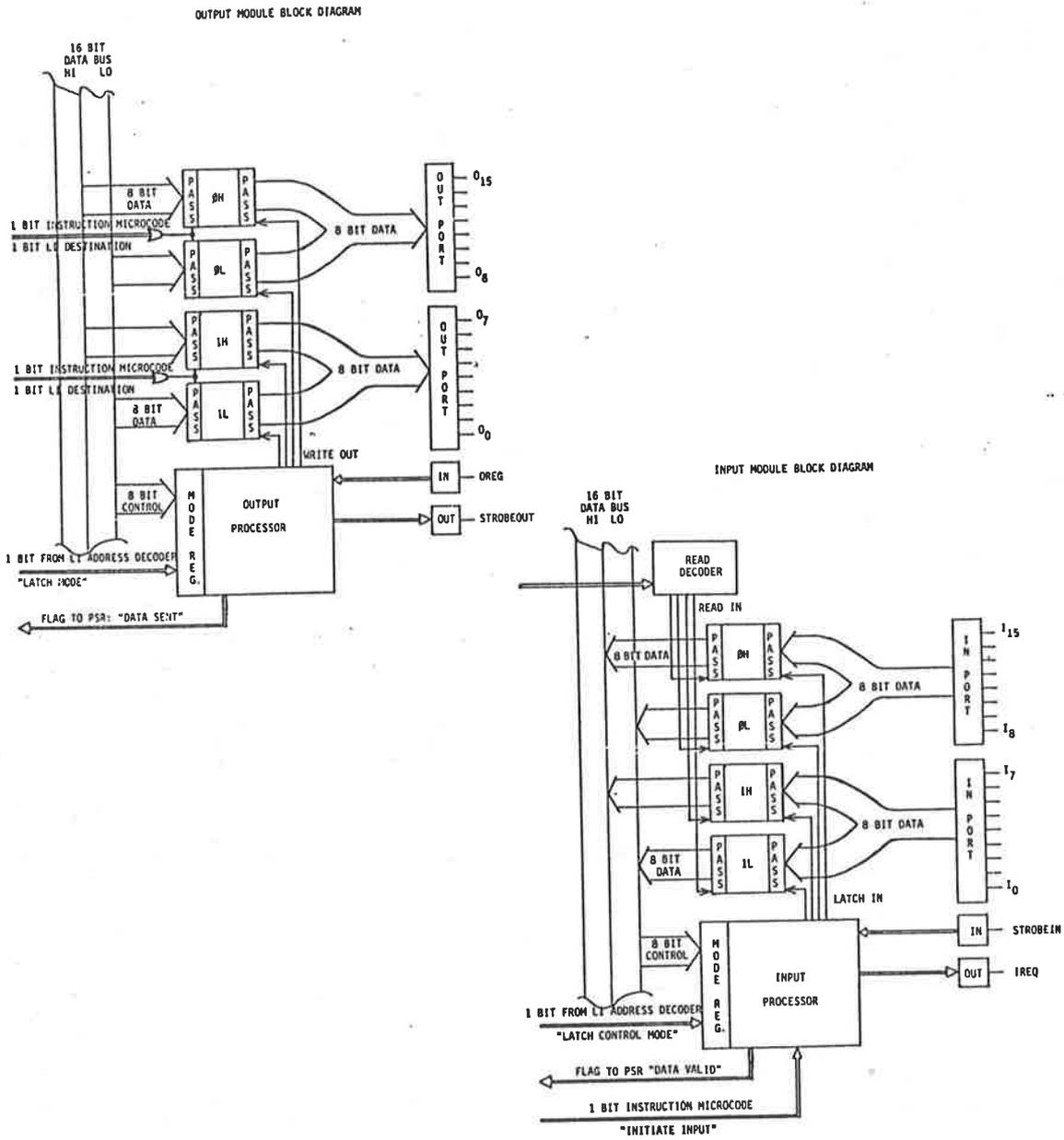


Figure 3.6: The Input and Output Processors.

Asynchronous operation is primarily intended for communication with foreign devices. Upon receipt of the input request signal the foreign device sets up the requested data at the input port, and generates a strobe pulse used by the input processor to latch that data into the TFB registers (two data bytes and two strobe pulses in succession are used in some transfer modes).

In synchronous modes of operation, the input processor sends the input request high on phase one of the clock, and data is transferred in on phase two and possibly on phase three (if the latter is required by the transfer mode). The double byte, dual port synchronous mode (mode 6 in the table) allows the transfer of two 16 bit words of input and two 16 bit words of output in one clock cycle.

3.2.7 The "Load Immediate" Bus Register.

The Load Immediate bus register is a 16 bit register sharing a 16 bit bus segment with the output processor. Its function is to move the "Load Immediate" data word from the Instruction Register onto the Ring Bus, providing a means of down-loading numeric constants and control words from the program store to the data-handling elements.

3.3 Control.

The TFB control has been specified primarily from the point of view of optimising the normal Program Execution mode for the previously described architecture of data handling elements. Any matters not directly related

to the Program Execution have generally been regarded as of subsidiary importance, with the notable exceptions of yield and testability considerations.

3.3.1 General Timing Strategy.

As mentioned previously, the TFB chip uses a three phase non-overlapping clock. The main reason for this choice was the initial uncertainty regarding the delays in transferring information around the Ring Bus, from one processing elements to another. As most of the data processing operations require two phase clocks, it was decided to add a third phase purely for global communications. All timing decisions were then referenced to this. Phase one is nominated as the communications phase, during which devices write to the Ring Bus, and a sufficient time is allowed for the bus segments to stabilise at their driven value. During phase two and three, data is latched from the bus and processed.

Using the three phase clocking scheme as a basis, the major timing steps were laid down as follows. To commence the instruction execution on phase one (*i.e.* the output of data to the bus), the data handling elements must latch their control lines no later than the beginning of that phase, and thus the instruction microcode must be decoded by the end of phase three. Allowing one phase for the combinational microcode decoding, the instruction must be fetched from memory by the end of phase two. Enabling the word-line to address the program memory at the beginning of phase two requires the decoding of the program pointer and memory bit-line precharges during

phase one. The pointer increment or load must then take place on phase three. Working forward from the instruction execution, if the execution is completed during phase three, then any completion or condition flags are set by the end of phase three, and so may be evaluated on the following phase one.

In typical programs, some instructions are *conditional*: that is, they are only executed if a particular condition is satisfied. In TFB this is achieved as follows. The instruction microcode word has certain bits set which dictate the particular condition to be assessed. On decoding during phase three, the condition bits are used to initialise the condition comparison circuitry. At the same time, the previous instruction is completing execution and setting flags in a Processor Status Register. During phase one, the conditional instruction begins execution, but the only actions taken during phase one are those which can be discarded without corrective action: this means in effect that no data is latched during phase one. Simultaneously, the condition is assessed by the comparison circuitry which sets a global NOP (No Operation) line if the condition is not satisfied, and clears the NOP line if it is. All elements whose operation is conditional on the result of the instruction condition may then assess the NOP line on phase two and complete execution or otherwise. There are some minor variations on the timing given here for certain elements, notably the program pointer load and the output processor, but the main criterion of discardability of the executed portion of the instruction remains in force universally.

This scheme allowed the effective elimination of the instruction pipeline. A major problem with executing programs in which non-sequential instruc-

tions ("jumps") are allowed under certain conditions, is avoiding the time loss in fetching the out-of-sequence instruction. In TFB, two program pointers are used, in conjunction with two parallel decoders. One of the pointers is associated with an automatic incrementation circuit, and is used to fetch the sequential instructions during normal program flow. The other pointer is loaded with the jump address every time a jump instruction is fetched, whether or not that instruction is executed. When a jump is executed, the jump pointer has been loaded with the address on the previous phase three, and decoding of the address occurs on phase one in parallel with the decoding of the next sequential address. On phase two, the jump address decoder is used to select the wordline, fetching the out-of-sequence instruction. The jump address is then written to the incrementing pointer, which increments it to produce the next sequential instruction address. If the jump instruction is conditional, the condition is evaluated on phase one, and thus the jump may execute correctly in phase two if the condition is satisfied. If the condition is not satisfied, the jump fails to execute, because the incrementing pointer is used in phase two to select the next fetched instruction, resulting in a continuation of sequential instruction execution.

3.3.2 Control Hierarchy.

The normal operational mode of TFB is the execution of a stored program. This requires the implementation of a Control Store, in which the program is stored, and an Execution Controller to sequence the fetching of instructions from the Control Store. These are illustrated in block diagram form

in Figure 3.8 and Figure 3.7 respectively. Below this level in the hierarchy are the decoders which derive the control lines for the various subsystems from the microcoded instruction word.

Although the normal program execution continues without reference to outside control signals (apart from the clocks), it was considered important to include a Single Step mode of operation to assist the debugging of applications software. This is implemented using the Execution Controller, but under the control of an external Single Step strobe rather than free-running. The switch between Single Step and Execute modes is controlled from external pins, and does not require the clocks to be stopped or gated. Rather, the Execution Controller utilises the NOP line to halt activity in the processor when the single step is complete.

The ability to halt processor activity in the Single Step mode allows this mode to be used for other purposes. The loading of the program into the Control Store is accomplished by shifting the instruction word in along a serial shift register chain into a Write Register, and then initiating a dedicated control circuit which generates the control signals to write the Write Register into the Control Store. Another use of the Single Step mode is for testing of the processor elements, both in their functional configurations and as distinct partitions of the elements.

To summarise, external control selects one of Single Step or Program Execute modes. If the latter is selected, the Execution Controller sequences the fetching and execution of the stored program, without further external action required. If the former is selected, then further external control is required to either cause the stored program to single step execute, or to load

| Instruction Type | Bit Numbers | | | | | |
|------------------|-------------|-------------------|----------------------|----------------------------|----------------------------|---------------------|
| | 1 | 2-4 | 5-10 | 11-18 | 19-34 | 35-38 |
| OP | OP | Condition Control | 6 x 1 bit Pass/Break | 4 x 2 bit T-Switch Control | 4 x 4 bit Data-Mem Control | 4 x 1 bit DMI Latch |
| LI | LI | | Program Flow | Data Handling | | |

| Instruction Type | Bit Numbers | | |
|------------------|--------------------------|----------------------|----------------|
| | 39-58 | 59-61 | 62-63 |
| OP | 4 x 5 bit ALU Control | Input Control | Output Control |
| LI | 16 bit LI Data + 4 spare | 5 bit LI Destination | |
| | Data Handling | | |

Table 3.4: Instruction Microcode

the program, or to use the test facilities. Various combinational decoders translate the instruction microcode into control line values in both modes.

3.3.3 The Execution Controller and Microcoding.

In both the Single Step and the Program Execution modes, the Execution Controller (Figure 3.7) accesses and executes a program of stored instructions. Part of each stored instruction dictates the flow of program execution, and the rest of the instruction dictates the actions required of the data handling elements. The instruction microcode template is illustrated in Table 3.4

The Execution Controller interprets the instructions' program control field, examines the appropriate flag information and generates signals to extract further stored instructions from the Control Store.

Each instruction may be one of two types: an OPeration (OP) instruc-

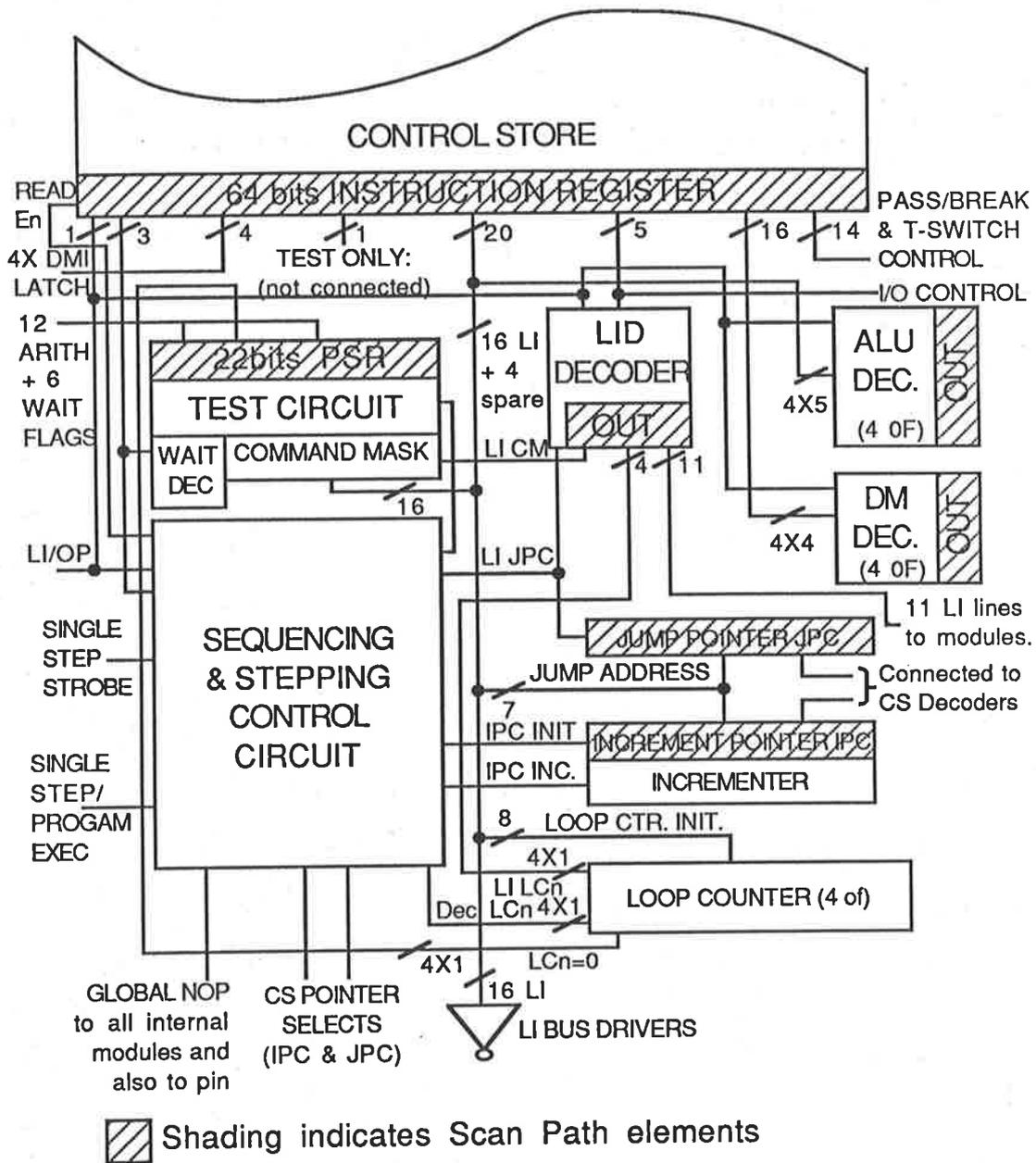


Figure 3.7: The Execution Controller.

tion in which the data handling elements manipulate or process data, or a Load Immediate (LI) instruction, in which a 16 bit field from the instruction microcode is loaded into one or more of a set of registers, either directly or via the Ring Bus. The instruction type is defined by the state of the single OP/LI bit in the microcode.

The execution of either instruction type can be made conditional upon the satisfaction of one of seven different types of expression, as well as having an unconditional form. The conditional expression is selected by the state of the 3 bit Condition Control field, and may be one of the following:

NULL unconditional execution,

TEST test the Processor Status Register (PSR) against the Conditional Mask (CM), discard the instruction on failure, execute the instruction on success,

WAITA wait until multiplier/divider A completes its most recently initiated process and returns a "COMPLETE" flag before executing,

WAITB same as WAITA, applied to multiplier/divider B,

WAITC same as WAITA, applied to multiplier/divider C,

WAITD same as WAITA, applied to multiplier/divider D,

WAITI wait until the most recently initiated input completes and returns a "COMPLETE" flag before executing, and

WAITO wait until the most recently initiated output completes and returns a "COMPLETE" flag before executing.

The TEST expression is used to test one of the 16 arithmetic flags stored in the Processor Status Register. This is implemented by comparing the PSR against the Command Mask, a register which has been pre-loaded with a bit pattern to select the flag of interest. The arithmetic flags comprise of 4 "NOT ZERO" flags, one for each of four Loop Counters, 8 "ZERO" flags, one from each of the accumulators and multiplier/dividers' Product Registers, and 4 "NOT POSITIVE" flags from the accumulators. Each Loop Counter is simply an 8 bit decrementing counter which is initialised by an LI command, and which decrements each time it is tested for zero and is found to be non-zero. The primary object of providing Loop Counters is to allow nested iterative loops to be programmed and executed easily. The remainder of the flags are provided to allow data-dependent operations to be programmed easily: these would typically be scaling operations or portions of adaptive algorithms.

The WAIT conditional instructions are included to allow the minimisation of stored instruction words, by eliminating NOP (NO Operation) instructions while waiting for the multiplier/dividers to finish their fixed period operations, and to ensure the safe completion of asynchronous data transfers to and from external devices. The PSR holds the six "COMPLETE" flags, as well as the 16 arithmetic flags. The conditional control field is decoded and a mask is generated to compare with the "COMPLETE" flags if a WAIT expression exists. If the selected "COMPLETE" flag is not set, the Execution controller sets the NOP line high and halts Control Store fetches until the flag is set. When this occurs, the conditional evaluation on the next phase one will return a true result, and the



conditional instruction will complete execution on the subsequent phases.

An LI instruction places a 16 bit word from the instruction microcode onto the Ring Bus, from whence it may be latched into most data registers. The same word may also be latched by various control registers within the Execution Controller (the Loop Counters, the Command Mask and the Jump Pointer). Use of the Ring Bus requires control over the Pass/Break and T-Switches, so the microcode fields for their control are interpreted identically for both LI and OP instructions. As the programmer may wish to load a constant from the Control Store to either the Delocalised Multiplier Inputs or to any of the Data Memories or their Pointer control circuits, their microcode fields are also interpreted identically in both cases.

Under an OP instruction, each ALU may execute one of thirty operations, and so the microcode for each consists of a fully encoded 5 bit field. However, under an LI instruction, each ALU is restricted to one of three possible accumulator latch operations, and so to minimise microcode space the four 5 bit fields interpreted as ALU control under OP instructions are used for the 16 bit LI data word (plus 4 spare bits) in an LI instruction.

The input and output processors require 3 bit and 2 bit fields respectively for an OP instruction, but under an LI instructions these control lines are not required. The Execution Controller registers, the accumulators, and the control registers of the Input and Output Processors may all be loaded with the LI data word, but no explicit method exists for selecting these registers. Instead, each of these registers may be addressed by the LI Destination, a 5 bit encoded address which is passed in an LI instruction by those fields used for the I/O control in an OP instruction.

The Execution Controller is implemented as a random logic circuit to generate all the sequencing controls, with another random logic circuit to generate the single step controls, a test circuit to compare the PSR with the CM and the decoded WAIT bits, a decoder for the LI Destination, and all of the associated registers: the Instruction Register, the Processor Status Register, the Conditional Mask, the Loop Counters, and the Incrementing Pointer and the Jump Pointer.

Each encoded ALU field is decoded by one of four identical combinational decoder. These are not considered part of the Execution Controller, and are located adjacent to the element they control. Similarly, there are four identical Data Memory field combinational decoders, one adjacent to and serving each of the Data Memories.

3.3.4 The Control Store.

The Control Store (Figure 3.8) is required to store up to 128 instructions, each of 63 bits. As discussed in Section 3.3.1, there are two separate pointers and address decoders capable of addressing the Control Store, one for sequential instruction fetches and one for out-of-sequence instruction fetches. The memory is written from a Write Register, which is a parallel output shift register, allowing instructions to be shifted in from the pins under external control. During program loading the processor must be in the Single Step mode, with all processing halted. The sequencing of control signals for the Write operation is handled by a dedicated controller, which is initiated by shifting in a control word and toggling an external pin. Read opera-

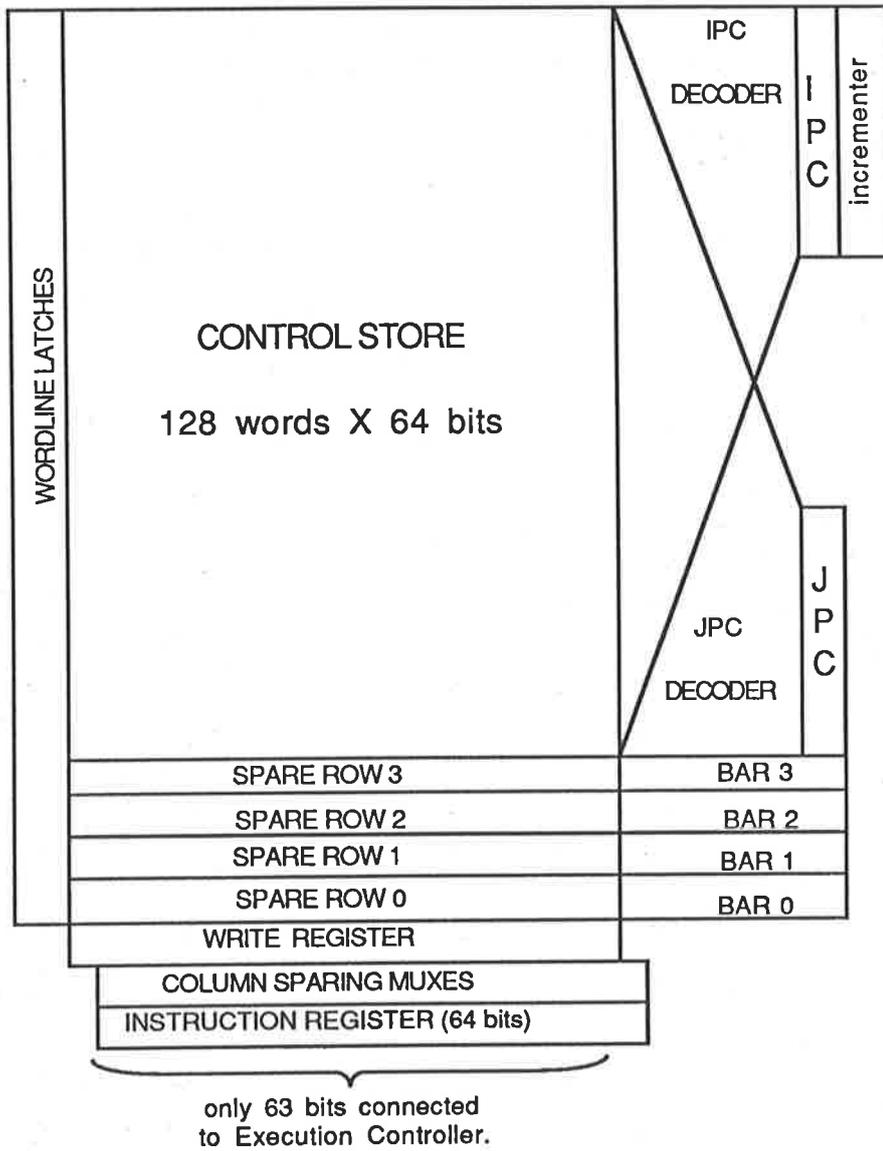


Figure 3.8: The Control Store.

tions are sequenced by the Execution Controller. The instruction fetched by a Read is latched into the Instruction Register, where it is held until overwritten by the next instruction. The outputs of the Instruction Register are passed to the Execution Controllers and the various combinational decoders.

Investigations into circuit yield and redundancy by Paul Franzon [46] indicated that a substantial improvement in circuit yield could be achieved if spare columns and rows were included in the Control Store and were used to replace defective rows and columns. It was shown that by providing one spare column and four spare rows, the overall chip yield could be improved by a factor of up to two. Increasing the number of spare rows beyond four gave little additional returns, and the complexity of the switching network for more than one spare column was considered prohibitive.

The spare column is implemented by simply adding a 64th column to the RAM array, and including a switching array between the memory output and the Instruction Register. The switching array consists of 64 two-position switches, each controlled by the value held in an associated latch. Each switch has as its output the corresponding bit position of the Instruction Register, and as inputs two adjacent column outputs. For instance, columns 1 and 2 are inputs to the first switch, the output of which is bit 1 of the IR. Similarly, columns 2 and 3 are multiplexed to bit 2 of the IR, and so on up to column 64. Note that this requires only 63 multiplexors and IR positions. However, to ease the problem of accessing all the Control Store storage cells to assess how to organise the sparing, a 64th multiplexor and IR bit position is implemented. The multiplexor is connected to column 64

at one input, but disconnected at the other. The 64th bit of the IR does not connect to any control line. To enable testing, the IR is designed to allow shift-register operation, and words are shifted into the Write register, written to memory, read out into the Instruction Register and shifted off chip for assessment. When a decision as to which column to discard has been made, a 64 bit string is shifted into the multiplexor latch chain, resetting the multiplexors so that bit positions 1 to 63 of the IR are each connected uniquely to a column.

The row sparing is implemented differently. The main RAM array comprises of 128 words, each addressed by a single word line, which can be driven by either of the two pointer address decoders. In addition, there are four extra rows of 64 cells, each addressed by one word line, and each of those word lines is uniquely addressed by a Bad Address Register (BAR). If a word in the main array is to be replaced by a spare, its address is written to one of the BARs, which is then enabled. While the pointer decoders are decoding their addresses, the enabled BARs compare their addresses with those in the pointers. If a pointer address is identical to one held in a BAR, then the BAR control circuit will set a control line which disables all word lines from that pointer for that cycle, and it will select its own wordline instead, thus effectively performing a sparing operation. The BARs are enabled by serially shifting in the required address plus an enable bit using a shift-register chain.

The features of the Control Store have been discussed briefly here. Those aspects of the design which are related to the testability of the Control Store will be reviewed in more detail in the next chapter.

3.4 Layout and Process Considerations.

As well as the considerations mentioned in the Design Overview, other factors came to light during the course of the design. It was realised that the overall size of the chip would be rather large, and thus not only must the layouts be space-efficient, but it would probably be necessary to employ a fine feature size process. The projected technology for final implementation is a two-level metal, twin-tub process with a 1.5 micron feature size.

The concept of the ring-connected data handling elements under the control of a single program execution unit fits well with the proposed layout of a central control block surrounded by the four identical data processing quadrants. However, the sheer size of the Control Store and the way it connects to the Execution Controller makes the routing of those lines which connect to the ALU and Data Memory decoders difficult if only a single metal layer is used. Furthermore, the ALU requires a 32 bit bus connected to its various registers, and the routing of this, the critical carry chains and the power and clock lines becomes difficult in single layer metal. Hence it was decided that TFB must be implemented in a process with two metal layers. This allows the global clock distribution to be carried in metal, reducing skewing problems. The 32 bit ALU bus can be run explicitly over the ALU without affecting the power and clock local connections, while the lines from the IR to the combinational decoders can be routed over the memory in second layer metal.

The circuit requires some 52 external pins for control, power and ground, clocks and data transfer during its normal function. It was decided to

attempt to limit the total number of pins to 64, to allow the chip to be packaged in a standard 64 pin DIP rather than use an expensive alternative such as a pin-grid array. This has been achieved with little adverse affect on the testability, most of the remaining pins being used to implement this effectively.

3.5 Conclusions.

In this chapter, an overview of the specification and design of the TFB processor have been given. For the sake of clarity and brevity much of the fine detail of the design has been omitted. However, it should be clear that the processor is a relatively complex device. The next major consideration is the verification of both the design and the implementation. While the design environment is unproven, the design team remain confident that simulation carried out on the designs will minimise, if not totally eliminate, the design errors. The brunt of the task is therefore to ensure that the end product, the silicon chip, can be tested without undue delay or expense, and as a result of that test can be said to be, with a high probability, in perfect functional order. This is addressed in the next chapter.

Chapter 4

Testing TFB.

In this chapter the methods and techniques discussed in Chapter 2 are examined with regards to their applicability to the task of testing the TFB chip described in Chapter 3. Each of the major partitions' test procedures is discussed in detail, after which the circuit modifications and additions for enhanced testability are considered. In conclusion a brief review of the complete chip test procedure is given. The full specification of the complete chip test is documented in Appendix A.

4.1 Test Constraints and Aims.

The task of test generation for this chip has been constrained by a number of factors. One of the obvious considerations is the design environment within which the chip is represented, and the availability or otherwise of certain types of tools within that environment.

Layout for the chip has been carried out using the VIVID[†] design environment [108]. VIVID consists of a number of programs which interface with a master technology file. The designer is removed from many of the process-dependent implementation details. In particular the layout task consists of placing circuit symbols on a virtual grid in which process spacing requirements are not considered. A CIF representation is obtained from this symbolic description through a hierarchical compaction process, in which the master technology file is invoked to define requisite spacings. As a direct consequence of the design process, no gate level description of the circuit, a pre-requisite for most automatic test pattern generation (ATPG) systems, is available. Furthermore, the design environment has neither the software tools nor access to sufficient computing resources to do ATPG for a chip of the size of TFB. Thus the specification of the chip test has been achieved entirely by manual test pattern generation, with the concepts of partitioning and "Design for Testability" being employed to reduce the complexity of the problem to manageable levels.

Another constraint is the lack of complex dedicated test hardware. This has meant that test procedures must generally be limited to synchronous functional tests, although certain asynchronous processes exist on the TFB chip. The most notable of these, the input and output handshaking processes, will have to be tested functionally over a representative range of allowable timing conditions, rather than measuring delays and propagation times to verify the modules.

[†]VIVID is a trademark of the Microelectronics Center of North Carolina.

The latter constraint is partly circumvented by the enforcement of a tenet of "Design for Testability": that is, to minimise the number of asynchronous operations and to make those operations observable. To this end dynamic CMOS logic was excluded from the design wherever possible, and static or pseudo-static latches were used for data storage, rather than edge-triggered devices or dynamic charge storage.

The use of synchronous functional tests on the chip is facilitated by the inclusion of the Single-Step Execution mode. In this mode, in the absence of an external Single Step Strobe signal the Execution Controller (Figure 3.7) and all data handling elements are held in an inactive state by the global NOP line. (The exceptions are any incomplete asynchronous processes, such as I/O handshaking or multiplication). This capability allows instructions to be fetched from the Control Store (Figure 3.8), decoded and executed one by one, and the effects of each individual instruction to be examined separately, as far as the observability of the process allows, without resorting to manipulation of the system clocks. One significant aspect of this mode is that, while the system is inactive, register or latch contents may be altered without causing unpredictable changes in the circuit state. In particular, scan paths and scan/set circuits may be operated, and the program instructions may be loaded into the Control Store.

The chip has been designed completely from scratch, with no pre-existing cell or module designs being utilised. Although the design environment includes architectural, logical and circuit simulators, these are unproven tools in that to this date no comprehensive comparisons have been made between the simulator results and real results from fabricated

circuits. Thus the intention is to fabricate various modules extracted from the circuit as separate chips, in order to ensure the functionality of the module designs prior to fabrication of the complete TFB chip. The success of the Multi Project Chip (MPC) concept, as typified by the AUSMPC series [94], encourages this form of design-and-test strategy. Other VLSI design teams have utilised this strategy, an example being the design and verification of the CSIRO speech processor chip [31]. Although the MPC constraints on process type availability, die size and pin-count may force the modules to be fabricated in a reduced configuration in some cases, the validity of the test can be maintained by reducing the number of identical parallel bit-slices in a module. The expected module configurations for this process are outlined in Appendix B.

The commitment to this strategy of verifying the module designs prior to the assembly of the complete chip essentially reduces the chip test to a production test, rather than a prototyping test, the inference being that the test result required is a simple go/nogo indication. However, there are certain other considerations which modify this conclusion. First, the Control Store RAM has been designed to have reconfigurable redundancies, to allow the replacement of defective memory elements. This "sparing" requires precise knowledge of the fault status of all memory elements, rather than a simple go/nogo indication. Furthermore this project is purely a research undertaking, and thus is somewhat financially constrained, so even if the chip under test is determined to have only some portions working correctly, it may still be of considerable use. Thus the aim of the chip test must be not only to determine whether faults exist, but also to determine which

portions of the chip can still function correctly in the presence of those faults. This has particular implications for the test structures designed into the chip, which will be discussed in later sections.

4.2 Fault Models.

The choice of fault models is influenced by a number of factors. First and foremost, as resources are not available to carry out ATPG and all test generation is therefore done manually, the fault models must be simple enough to use in this fashion. This precludes global transistor level modelling of faults, since the complexity becomes too great to manage manually, and in general it precludes gate level modelling unless certain conditions hold. The other criteria involve the actual design and layout of the particular circuit.

A general sequential circuit can be considered to be made up of single bit latches, with wiring paths and combinational circuits interposed between the latches. Such a circuit may be said to be fault-free if the latches may be shown to be fault-free, and, in separate tests where required, the combinational circuits and the wiring is shown to be fault-free. This assumption of the separability of the tests lies at the heart of test methodologies such as the scan path.

In choosing fault models for general sequential circuitry, the various circuit elements may be modelled differently, if that is of any advantage. The advantage only accrues if there exists within the circuit sufficient structure to allow this detailed consideration to be done in an efficient manner.

Specifically, if a circuit consists of numerous instances of a single cell in parallel, then the time spent analysing one in detail can be amortised against the other identical cells. This has been done in generating the test patterns for most of the circuitry in this chip test. The highly structured design style has made possible the manual generation of patterns for specific leaf cells of the layouts.

The latches and registers throughout the chip generally exist as bitslices grouped together physically in larger structures. As these groups are similar in all major respects they are all treated similarly with regard to fault modelling. The latches and registers are almost universally of the CMOS master-slave type, as illustrated in Figure 4.5, and in modelling this cell it was considered that each such register or latch cell could take the individual fault states of Stuck-At-1 (SA1) or Stuck-At-0 (SA0).

However, because of the physical proximity of adjacent cells in a multiple bit register, it was also considered that adjacent cells may, via shorting or charge leakage mechanisms, influence the values in their neighbours. Because the cells are of a pseudo-static design and are therefore relatively insensitive to stray charge fluctuations, it was considered that they are unlikely to be subject to faults of a solely dynamic nature, occurring only on transitions, but they may be subject to static adjacent cell pattern interference (API), where the fault may only manifest itself when a specific value is held in the adjacent cell, and another specific value is required in the cell under consideration. This can be tested for by subjecting the cell pair to the four binary patterns $(a,b) = (1,1), (1,0), (0,1)$ and $(0,0)$.

If these binary pairs are applied in parallel to all the bit slices of a large

register, then the applied patterns may be represented by the hexadecimal notation $\langle F \rangle$, $\langle A \rangle$, $\langle 5 \rangle$ and $\langle 0 \rangle$, respectively. Patterns of this form will be referred to throughout this work as an API test set. Note that in some cases the applied data at a point removed from the cell under test may not be of this form: account must be taken of the interposed logic cells on the data path under consideration, as well as the physical layout of the cells. For instance, the Data Memory buffers are arranged so that adjacent cells belong to different buffers, and thus merely writing $\langle AAAA \rangle$ to the cells of one buffer does not necessarily test the cells for API faults. In general, however, most of the registers are of a simple form, such that the API test set will be of the form shown above.

The memory cells form a special restricted sub-class of register cells, and as such have a restricted fault set. This arises specifically from the memory array layout, and it will be discussed more fully in the section on Control Store testing.

The combinational logic cells are assumed to have allowed fault states of SA1 and SA0. The "Stuck-Open" sequential fault is not considered for two reasons. First, there is no generally applicable test set that guarantees the detection of Stuck-Open faults. Secondly, the prevailing industry practise is to use only Single Stuck-At (SSA) fault models, and the experience with this indicates that Stuck-Open faults do not escape a "good" SSA test in any great numbers: whether a test is good or not is *a priori*, but there is some correlation with tests that do more than just apply a bare minimum of patterns. The test here is in that latter category, due to the efforts made to maintain a high level of fault location.

Rather than attempt to model the combinational cells at a gate level, they are usually treated by applying pseudo-exhaustive test sets. This relies on knowing which outputs relate to which inputs, but fault models are not explicitly considered, it sufficing to note that the postulated faults cannot change the structure of the circuit.

In many cases the combinational logic is tested pseudo-exhaustively simply as a result of applying API sets to the registers on the outputs of the circuits, so separate tests specifically for the combinational circuits are not required. In some cases, such as the single bit adder cells, a test set is specifically applied to ensure that the adder is pseudo-exhaustively tested. In specifying the tests, the structure of the combinational logic is taken closely into account. For instance, the global NOP line is combined with the ALU and DM Decoders' outputs after the output latches, so there is no need to test the effect of the NOP for every state of every output line, but simply to ensure that the NOP line will affect that line correctly, given the output latch is at one or other value.

The wiring fault set is assumed to consist of SAO, SA1, bridging and opens. The API test set is usually applied to parallel data paths, and this will detect the first three fault types without a doubt. Opens in the wiring are generally detectable by cycling the wire through the voltage range, although in the Ringbus test the extra precaution of reverse-charging the bus is taken to aid in break detection.

4.3 Test Strategy Overview.

One of the immediately obvious features of the TFB chip is its modularity: each of the data handling elements interact only via the Ring-Bus. Furthermore, as the control sections are to a large extent modular and the control of the chip is hierarchical, the control sections may in many cases be regarded as independent of each other.

This natural partitioning of the circuit into modules with limited interaction is a major step in reducing the complexity of the testing task. The partitioning has been further aided by the introduction of design constraints requiring that signals from the control to the data handling modules be passed as logic levels wherever possible, rather than as timed signals, with the modules' internal control circuits imposing the timing.

Another aspect of the partitioning is that the data handling elements often consist of multiple copies of identical bit-slice cells handling parallel data streams under the control of a single circuit. The multiple bit-slices often have little interaction between themselves, thus simplifying the test procedures. This can be of considerable use at the module or partition testing stage.

With regards to the various natural partitions, some of these are inherently easily testable, while others present difficulties arising from their size, complexity and lack of accessibility. The remainder of this section seeks to differentiate between those partitions which are hard to test and those which are easy, and to point out specific reasons for this differentiation in each case.

Consider the combinational decoders which translate the instruction microcode into control lines required by the ALUs and Data Memories (Figure 3.7). These accept microcode directly from the Instruction Register, and thus, provided the facility to load and execute a program can be verified, controlling the inputs to the decoders is easy. The outputs of the decoders, latches holding logic levels at the inputs to the data handling modules, are not directly observable from any external ports of the system. However, the structure of the outputs suggests that a scan path could be readily implemented to include the decoder outputs, thereby allowing serial output from the decoders to an external test evaluator. Another possibility is to reconfigure the output latches as a linear feedback shift register to compress the results, but this still requires scanout capability or alternatively an on board signature comparison. The independence of the decoders allows their parallel testing. The small size of the input code fields (4 bits for each Data Memory decoder and 5 bits for each ALU decoder) facilitates the use of an exhaustive test set.

One advantage of making the output decoders part of a scan path is that the control line inputs into the data handling modules are settable, even if the decoders or other portions of the control hierarchy are inoperable. This is in accordance with the idea of extracting as much performance information as possible from a chip, even if it is partially defective.

It is thus possible to apply the normal functional control inputs to the data handling elements. This will suffice in some cases to provide a full test of the modules' functionality. For example, the Data Memories (Figure 3.3) may be forced to read and write from a specific address under normal

instruction execution. This may be used to test the address decoders, the actual RAM array and the data buffers for each Data Memory. When this is complete, the pointer control circuitry can be tested by preloading the memories with flag words that are unique to the address in which they reside, then applying instructions which exercise the pointer control and comparing the output flag word to that expected.

The Input and Output Processors (Figure 3.6) contain data registers which can be considered as Iterative Logic Arrays (ILAs), with no intercell dependence, and hence may be tested in parallel. The controllers are simple finite state machines, and the dependency of the register contents on the control is fairly simple, so providing the instruction microcode to the controllers can be verified, a functional test should suffice to verify the complete processors (subject to the previously mentioned *caveat* regarding the asynchronous modes).

The Ring-Bus may be tested using the Output Processor to export the results off chip. The stimulus could come from the Load Immediate bus register, or for simple patterns the T-switches' zero fill/sign extend capability could be used.

The Adder/Subtractor/Shifter/Accumulator (ASSA) partitions may be subdivided even further for the purposes of testing: the Accumulator/Shifters can be considered to be largely independent of the Adder/Subtractors for certain portions of the testing, reducing the test complexity. Furthermore, there is only limited intercell dependence between the bit-slices of the partitions. Given controllability and observability of the bus and associated switches, and controllability of the control

lines into the partition, the ASSA should be testable as a functional unit, simply by applying each possible instruction together with a selected subset of data appropriate to that instruction.

The above-mentioned partitions all appear relatively easy to test. On the other hand, testing the remaining two partitions, the multiplier/dividers and the Control Core, poses some problems.

Consider the multiplier/dividers (Figure 3.4): with two 16 bit inputs, and a multiplication cycle of nine clock periods, exhaustive testing of these partitions is out of the question, without even considering the divider test. "Pseudo-exhaustive" testing is rendered difficult by the dependencies between bit-slices, notably the shift and carry chains and the shift/invert cells. Another major difficulty arises from the "start and let run" style of operation, in which the multiplier/divider internal control circuits control the entire 9 clock period multiplication or 17 clock period division without reference to any outside control, other than the system clocks, once operation is initiated. This in many ways can be regarded as an asynchronous operation since it is difficult to reference any portion of the operation to any given system event, and the intermediate results are not readily observable.

One feasible method of testing this module is to further partition it, gaining controllability and observability of internal nodes. This would require the addition of some circuit components purely for testing purposes, and scan path elements or scan/set elements are possible ways of achieving the required controllability and observability. Another possibility is to alter the control circuit behaviour in a test mode to produce synchronous, rather than asynchronous, operations: this means reducing the multiply or divide

cycle for a test instruction to a single adder operation. This allows the pseudo-exhaustive, or path-sensitisation approach to testing the module.

Lack of controllability and observability is again a problem in the Control Store area (Figure 3.8). For the normal function of this partition, external access is required to load the program instructions into the Control Store, initialise column sparing by setting multiplex position latches and initialise the "Bad Address" registers used for row sparing. Other inputs to this partition are the two pointer registers, but these do not need to be directly accessible from the external pins, being primarily loaded by the Execution Controller. The observability of the partition's elements is a more serious problem, for no direct external outputs are generated by the partition in normal function, and yet it is necessary to have detailed information on the fault status of all cells in the partition in order to perform a sensible row and column replacement for yield optimisation.

Given the lack of external observability of the partition in its functional configuration, some effort was directed to designing a totally built-in test and reconfiguration circuit for the Control Store (discussed in Appendix D). Based on the premise of a limited fault model for the RAM array cells, it simply gives a go/nogo external indication upon completion of testing. However, the penalties are a large increase in the area due to the testing overhead, no flexibility in the test routine and a total lack of information on fault location. The first and last points are considered compelling arguments against the adoption of this scheme.

Instead, the problems of observability and controllability can be attacked by incorporating many of the existing registers and latches into

scan paths or scan/set circuits. The area and complexity overhead in this type of scheme is much less than for a totally built-in tester and the test format for the RAM is only limited by the scan time between successive patterns and the scope of the external test equipment. The time penalty incurred by scanning patterns into and off the chip is partially offset by the use of a minimum size test set, rather than one tailored to allow a simple automatic generation scheme. Further, the extra observability allows a greatly enhanced fault location capability.

Similarly, the Execution Controller, although indirectly controllable via the program instructions, has no directly observable output (Figure 3.7). By incorporating into scan paths the various registers which hold the inputs and outputs of the Execution Controller, the testing of this partition is facilitated. These include the Instruction Register (IR), the Processor Status Register (PSR) and the Control Store Pointer registers (IPC and JPC).

From the foregoing observations, it can be concluded that the introduction of one or more scan paths through appropriate elements of the circuit may suffice to make the circuit testable, although other test circuits may prove to be the better solution in some cases. In the following sections the testing of each partition is discussed, with regards to both the circuitry and the test procedure, and then the test circuitry for the various partitions is addressed as a complete system.

4.4 The Control Core.

The Control Core is the section of the chip involved in the storage and interpretation of the program instructions, and the sequencing and timing of the execution of those instructions. It includes the Control Store and the peripheral circuitry to allow loading, testing and reconfiguring (Figure 3.8), the Execution Controller, the Load Immediate Destination (LID) Decoder, and the connections to the external controlling pins such as Single Step/Program Execute, Single Step Strobe, and Write (Figure 3.7). The scan paths through the ALU and Data Memory (DM) Decoder outputs, although strictly not a part of this core section, are also tested at this time for convenience.

The Control Store is required to be reconfigurable to allow yield improvement through the deselection of rows or a column containing faults. The mechanisms, discussed in the previous chapter, involve the use of an extra column and four extra rows of memory cells, with address bit-comparison circuits effecting the row sparing and a row of multiplexors effecting the column sparing. The data to set the multiplex row latches and to fill the BARs is shifted in serially.

The Write operation is initiated from an external pin latched by the system clocks with the latched signal modifying the internal operations to allow the Write Register contents to be written into the memory array. The address at which the data is written may be determined directly from external pins by shifting in Control Input bits which select the pointer to be used in the Write operation. The instructions to be loaded into the

CS are shifted serially into the Write Register, which is separate from the Instruction Register, partly due to the multiplexor interposed between the base of the memory and the IR, and partly because testing is facilitated by having separate registers.

From the foregoing it can be seen that the operational requirements of the chip require a certain amount of serially shiftable registers to input data to various elements of the control core. The use of shift registers economises on external pins, and complex internal distribution circuitry is not required. A penalty for adopting the serial input approach is the comparatively long time required to input the data, compared to a parallel load, but the inputs serially loaded under this scheme are only utilised in testing and initialisation of the circuit, rather than in program execution, and thus the time penalty is more tolerable. Summarising the operational requisites for serial input, there are 64 bits of Write Register, four of the BARs, each with 7 bit addresses and an Enable bit, the 6 bits of Control Input to select which pointer to write with, and 64 bits of multiplex control.

The testability of control core elements requires other points to be controllable or observable, or both. In particular, the IR needs to be observable to allow testing of the memory array, and the ease of conversion of the master-slave register cells of the IR into scannable cells suggests this convenient solution (Figures 4.5,4.4). Furthermore, the electronic sparing scheme requires detailed knowledge of the failures in the memory and its address decoders. Partial or complete row failures may arise from decoder faults or faults in the wordlines themselves, and may be difficult to isolate to a single row by examination of the IR contents. If we introduce master-

slave latch pairs on the end of each CS wordline, written every time a Write operation occurs, and these are joined to form a scan path, then the address decoders and word lines are directly testable, particularly if the pointers IPC and JPC are also controllable and observable. Another consideration is that the restricted fault model used for the memory cells allows a simple test word format: all zeros or all ones. This pattern may be generated in the Write Register by making it clearable and settable under the control of two further Control Input bits, Write Register Set and Write Register Clear.

The controllability and observability of the pointers is also important in considerations of the testability of the Execution Controller, for which the incrementation of IPC and the parallel load of JPC must be verified. The arithmetic and task completion flags, which are used by the Execution Controller to determine the program flow under a conditional instruction, are held in the Processor Status Register (PSR). Another important register group are the output latches of the LID Decoder. Including these registers as part of a scan path allows direct control and observation, thus facilitating the testing of the Execution Controller.

Implementing these extensions to the scan path segments required for operational purposes adds 64 bits of IR, two more 1 bit Control Input bits, two 7 bit pointers, 22 PSR bits, 18 bits of LID Decoder output and a 132 bit wordline latch segment. There are two pairs of control core scan paths, and two pairs of decoder scan paths, each pair of paths being multiplexed to an input and an output pin, with the clocking organised so that individual paths may be scanned, or a pair of control core paths or

decoder scan paths may be scanned out simultaneously (on different pins). The complete details of the scan paths' implementation and organisation are discussed further in Section 4.12.1: for this discussion it suffices to assume that the elements included on scan paths can be both set and read at will.

One final addition is made to the chip to enhance the control core testability. The global NOP line, which controls the instruction execution at the distributed modules, and responds to the result of the conditional expressions, is passed off-chip via one pin. This allows a more immediate assessment of the results of conditional expression evaluations, enhances observability of the Single Step method of operation, and provides a method of flagging the operation of internal program execution before the Output Processor is verified.

The control core test is the first major digital test that can sensibly be performed on the chip. The first task is to verify that switching the chip into Single Step mode forces the NOP line high, disabling the normal program data latching processes and thereby setting up the conditions under which the scan paths may be operated. When this is established, the scan paths, including those through the DM and ALU Decoders, are scanned through to verify their ability to hold and pass zeros and ones. Given that all of the scan path elements are registers, and thus are structured as rows of identical cells, the possibility exists that, given a fault in one cell, the value in the adjacent cell may be corrupted by the value in the faulty cell: this shall be referred to as an Adjacent Pattern Interference (API) fault. As all storage nodes in this chip have been designed as pseudo-static latches, and are fairly

robust in terms of sensitivity to stray charge, it is considered unnecessary to test for dynamic API faults, and only static patterns will be considered. To detect this type of fault it is simply a matter of writing adjacent cells with all possible permutations of zero and one. Clearly any pattern testing for all static API faults will also detect any Stuck-At (SA) faults. Thus patterns consisting of repetitions of the group of bits represented by hexadecimal 0FA can be used as a detection test set for API and SA faults in the scan paths. These are scanned into the scan paths until the first complete 0FA group emerges from the scan out pin. After this the scan is continued with a different identifiable pattern until that pattern scans out also, and the scan count may be used to verify that the scan path selector circuit is working correctly, as the control core scan path lengths are all different. The first part of the scan test is done as individual scans to verify the scan capability of each path, while the second may be done with pairs of paths in parallel to reduce test time, and to verify the parallel scan capability.

The next test is to partially verify the Write operation: the latching of the external signal, the latching of the memory wordlines, the selection of pointers by the Control Inputs, and the post-Write clearing of the latched Write signal. The Write pin is set once, and the values left in the Control Input latches by the preceding test select pointers. The wordline latch scan path is scanned out and examined, and should show only a single write operation, albeit using several pointers simultaneously.

Having verified that the wordline latches work, they may now be used in conjunction with the scan in capability of IPC and JPC and the Write operation to test the address decoders of IPC and JPC, and the main CS

array wordlines. The Control Inputs are serially filled with a select for IPC or JPC, and the required address in IPC or JPC respectively, the Write pin is pulsed and then the wordline latch path is scanned out to check for correct and unique decoding and intact wordlines and drivers. Both IPC and JPC are used to select each address, in order to exhaustively test each decoder. By using the Write Register Clear Control Input bit in every case, this test also serves to initialise the main CS array for the subsequent cell test.

The next test is to verify the operation of the BARs. Their register capability has been demonstrated, but it remains to verify that each will select its own wordline correctly, and that each is capable of deselecting or over-riding the IPC and JPC decoders outputs when the BAR is selected. First each BAR is selected individually by simultaneously setting its Control Input bit and that of either IPC or JPC within a Write instruction, and examining the resultant wordline latch scan. The selection of the BAR should overwrite both IPC and JPC, and only the BAR wordline should be set. Next each BAR is individually checked for its ability to match its contents with IPC and to over-ride the pointer decoder output. This test is then repeated with JPC. In each case the result is determined by examining the wordline latch scan. Having demonstrated that each BAR individually can over-ride the IPC and JPC decoders, some time may be saved by continuing the match and mismatch tests of the BARS in parallel, and determining the results from the wordline latches corresponding to the BARs. The BARs use bitwise comparisons to compute matches with the pointers, and so to test the comparators fully for SA and API faults

the match test set comprises of the all ones pattern, the all zeros pattern, and the two patterns of alternate zeros and ones (in hexadecimal notation F,0,5,A). The mismatch test set consists of all pairs of numbers such that one number is all zeros or all ones and the other only differs in a single bit position, plus the multiple mismatch case of one number all zeros and the other all ones. Each pair in the mismatch set is applied twice for each combination of BAR and pointer, with the numbers being swapped over to opposite registers the second time. This completes the BAR addressing test. At this stage any row faults should have been detected, and steps may be taken to avoid those addresses in the subsequent tests. Note that in this test also the Write Register Clear bit is utilised to complete the initialisation of the memory cells to all zero.

The next test checks for column faults: faulty or stuck-at precharge or write drivers, broken BIT or BITBAR lines, stuck-at sense-amplifiers. To do this ones and zeros are written to the memory and then read out into the IR, from whence they can be scanned out and examined. This is achieved by scanning into the Control Inputs an IPC select bit, and an IPC initial address, while the Write Register is filled with the appropriate pattern, and the PSR with flag test patterns. The Write pin is then pulsed to load the patterns into memory. Then the Single Step Strobe is pulsed to read the pattern from CS into the IR, and thence into the Execution Controller.

As is discussed in relation to the CS cell test below, the memory is not considered likely to have API faults between adjacent memory cells of any particular row. However this test sequence uses the alternating ones and zeros patterns to test the switching action of the multiplexor row.

As words are read from the CS by using the Single Step Strobe to initiate a single execution cycle, this test may also be used to partially test the Execution Controller's operation. In particular, as the condition control field of the instruction words read will be known, and the PSR may be initialised prior to the execution, examination of the NOP external pin will indicate correct operation of the conditional control circuitry, or otherwise. The condition control states examined here are the UNCONDITIONAL state (all zeros) with the PSR set to all ones, the TEST (all ones) state with the PSR set to all zeros, and two WAIT states (010 and 101) with the PSR set to patterns of all ones except for a single zero in the bit corresponding to the WAIT state, and all zeros except for a single one in the bit corresponding to the WAIT state. It makes no sense to test for specific passes in the TEST case, as the Command Mask has not yet been defined, so those tests will have to wait until the LID Decoder has been verified.

Furthermore, as the IPC is incremented as a consequence of the Single Step Execution, and the actual address in CS to which the test pattern is written is not important, by presetting the IPC address prior to the increment we can pseudo-exhaustively test the 7 bit incrementer attached to the IPC: the result is examined simply by scanning out the IPC after the increment has occurred. After the column and multiplexor tests have all been completed, a final three Single Step Executions are carried out, without Writes, to complete the IPC incrementer test.

The next step is to test the actual memory cells throughout the complete array for cell faults. Consider the structure of the memory: it consists of a

regular array of identical cells, with separation between the columns of cells to route power and ground metal rails, and separation between the rows to route the wordlines. Now it has been observed [55] that the most prevalent failure mechanisms in MOS technologies involve shorts and opens in the metal layers of circuits. The memory layout is such that, if a metallisation bridging fault occurs in a cell, then it may be purely contained within that cell, or it may connect the cell to either of the adjacent neighbour cells within the column, but it is not likely to connect to the row neighbour cells as the power and ground rails are interposed, and the result would be a Stuck-at (SA) fault. Where the neighbouring cells are bridged together, the fault may be SA, but more generally may be a data-dependent fault, in which the value in one cell may affect the value in the other. Inter-cell shorts on other layers are considered unlikely, due to the separation regions for the rails and wordlines. The memory cells are robust 6 transistor pseudo-static cells [136], and are unlikely to be subject to dynamic API faults arising from charge-leakage and similar mechanisms, so static faults due to bridging are all that will be considered. The resultant fault model for the cells is then a simple one: they may be SA-faulted, or they may have a static API fault involving one or other (but not both) of the nearest column neighbour cells.

This simplified fault model makes the test patterns rather simple: it suffices to verify that each adjacent column cell pair can be correctly written with all permutations of ones and zeros. Intra-row interactions need not be tested for, and thus all cells within a row may be tested with the same pattern at the same time, so the test patterns applied to the memory array

are words of all ones or all zeros. The well-known MARCH memory test [22, page 157] adequately generates all these patterns and in fact tests for transition faults as well, so in this case a subset of the MARCH test is used, in which the transitions are marched up the array, but the march back down again is omitted.

At the conclusion of the last test section, some addresses were left filled with non-zero words, and these are first cleared, so the array is once again filled everywhere with zeros. Then the cell test is carried out iteratively by first checking the row contents to be zeros, then writing the row to ones, confirming that write with a read, checking that the row below has not been altered by the write, and then (commencing the next cycle) checking that the row above is all zero after the write. After marching once up the array, the array is filled everywhere with ones, and the procedure is repeated with the transition taking the rows from all ones to all zeros. Address selection is achieved by scanning in IPC contents and where necessary BAR contents and Enables, as well as other Control Inputs, and this is done concurrently with the scan out to check the IR contents. A CS read occurs every time the Single Step Execution is activated by the Strobe pin, and the Write is initiated by the Write pin, and controlled by the Control Inputs scan path values.

After the cell test, enough information has been derived to allow an optimal decision on reconfiguration of the CS. This is achieved simply by scanning in the BAR contents and Enables as required to perform row sparing, and by scanning in 64 bits into the latch chain setting the multiplex row to the correct positions for the column sparing operation.

The memory having been verified, it becomes possible to test the rest of the control core with reasonable ease. Earlier test patterns have partially exercised the conditional evaluation circuits, but there remain some states to be tested. Also pointer selections need examination, as do the LID Decoder outputs, the JPC Load Immediate, and the Loop Counters' loads, flags and decrements.

The conditional evaluation circuits, as noted previously, are independent of the instruction type (LI or Op), and so LI instructions may be used to simultaneously exercise the LID Decoder and the conditional evaluation circuit. The PSR is scan-settable, and so all flags are controllable, and the NOP line is externally observable and may be used to determine the result of the conditional evaluation.

The first part of this test section examines the behaviour of the LI JPC instruction: this is crucial for program looping. The instruction and PSR values are scanned in on one path, while Control Inputs to set up the pointers is scanned in on the other control scan path. A Write operation loads the instruction to CS, and a subsequent Single Step Strobe causes the instruction to be fetched, a conditional expression to be evaluated, and the NOP line flags the result. Scanning out the IPC and JPC allows the effect of the LI JPC instruction to be examined. This test is carried out in four variant forms. The first two test the conditional evaluation for an unconditional instruction, while performing JPC loads with the two alternating one and zero patterns (2A and 55 in hexadecimal). These patterns are chosen to detect bridging faults in the LI lines to the pointers. For a successful LI JPC, the values of both IPC and JPC will be the loaded value. There is

a test with an Operation instruction, in which all bit values are identical except for the LI/Op bit, to verify that pointer loads do not occur on Operations, and here neither IPC nor JPC will be loaded. Finally there is an unsuccessful LI JPC, identical to a successful load except that the condition control field is altered to a TEST state, and this should verify that the JPC was loaded, but due to the conditional evaluation returning a FALSE value the JPC was not selected and the IPC was not loaded with the new value. This completes the testing of the LI JPC instruction in Single Step Mode: it is still necessary to ensure that the jump executes correctly in Program Execution Mode, as the pointer selection has not been thoroughly examined at this stage.

Next there is a concurrent test of the condition control WAIT state (CC field = 001) and the LI Loopcounter 0 instruction. First a failing LI is executed, to verify the WAIT bit and to show that the flag (Loop0 non-zero) is not set, then a successful LI is executed, completing the testing of the (CC = 001) state of condition control, verifying that the Loopcounter 0 flag is set to zero correctly, and that the appropriate LID Decoder line is set. Finally an unconditional Op instruction, with all except the LI/Op bit identical to the successful load, is executed to verify that the LID Decoder outputs are not activated by an Op. This test sequence is then repeated three further times, with the condition control fields set to (110), (100) and (011), and the LID selecting Loopcounters 1, 2 and 3 respectively. This completes the testing of the condition control circuits except for the TEST state, which requires prior verification of the LI Command Mask (CM) instruction.

Other LID Decoder output lines are the Accumulator High and Low Byte Latch lines, used to load the accumulators with a LI data word because the ALU decoders are disabled under a LI instruction. There are eight lines, and thirteen allowed states setting these lines: four each of High byte WRite ACCumulator (HWRACC), Low byte WRite ACCumulator (LWRACC), and Both bytes WRite ACCumulator (BWRACC), and one of Both bytes ALl ACCumulators (BALAC). Each of the thirteen states in turn is tested by executing first an unconditional LI instruction to verify that the LID Decoder output is set correctly, and then a LI with a failing TEST condition to verify that the NOP will disable the outputs. The structure of the LID Decoder is such that the LI/Op bit gates the outputs of the decoder, rather than the internal states, and so the effect of the Op instruction on these eight Decoder outputs can be tested using an Op instruction which is identical to the unconditional LI BALAC instruction (in which all eight lines are activated), except for the LI/Op.

Similarly, the I/O Control Latch line, the Output Register 0 Latch line and the Output Register 1 Latch line may each be tested using three instructions in the same form as the Accumulator line test set above, with the appropriate LID field. The same strategy may be used to verify the LI CM instruction, and the successful load can be used to load the Command Mask with the mask to test the first of the arithmetic flags. TEST conditional LI CM instructions are then used to check that the CM is loaded with the correct mask, by first setting all except the first arithmetic flag to one to get a failing condition, and then setting all except the first arithmetic flag to zero to get a TRUE condition. The results of the TEST evaluations are

flagged externally by the NOP pin. The successful LI CM is used to load the CM with the mask for the second arithmetic flag, and the process is reiterated until all sixteen arithmetic flags have been tested, and the TEST form of condition control has been verified for all single flag cases. The process of loading the CM with single set bits effectively tests the register for API faults as well as SA faults. A further TEST instruction tests the case of multiple flag tests, by loading the CM with all ones and testing against a PSR holding some ones and some zeros. Using this successful TEST to load an all zero CM, the last TEST instruction provides a fail regardless of the PSR state. This concludes the verification of the TEST conditional expressions, and of the Conditional Mask. The conditional evaluation circuits have now been verified completely.

The remaining tests use Program Execution Mode to verify certain aspects of the control operation. Single Step Execution always utilises the IPC to access the CS, even in the case of program jumps, because this mode removes the fetch/decode/execute overlaps inherent in Program Execution mode. To ensure that the pointers are selected correctly in the case of jumps, a small program, including unsuccessful TEST and unconditional jumps, is loaded and executed. By observing the NOP pin it is possible to deduce the periodicity of the program looping and hence determine whether the unconditional jump is executing correctly, or if an incorrect jump is arising from the failing TEST instruction, or if no jumps are occurring whatsoever. If the program cycles correctly and the correct pattern is observed at the NOP pin, the pointer selection is verified.

The Loopcounters have been tested for the ability to load the all zero

pattern, but it still remains to test the cells for SA and API faults, and to test the circuit decrement capability. This is easily achieved in the Program Execute Mode by loading the CM with the mask for the (Loopcounter non-zero) flag, loading the Loopcounter with the maximum size count (FF hexadecimal) and then executing a TEST LI JPC instruction, which is effectively a "jump not zero" instruction, with the JPC load pointing to the TEST instruction. The Loopcounters have been designed to automatically post-decrement on test, and thus the effect of this program segment is to cause the execution of a loop, decrementing the Loopcounter and holding NOP low, until Loopcounter equals zero and then the NOP is set high, flagging the finish of the loop to the pins. By counting the number of execution cycles that the NOP pin stays low, the number of decrements may be deduced and thus the load FF capability is verified for this Loopcounter, and the decrements are exhaustively tested. If this procedure is repeated with the initial loaded counts being the patterns of alternating ones and zeros (AA and 55 hexadecimal), then the API fault detection set has been loaded into the register, and we can deduce that there are no API faults on the input lines or within the register. By changing two instructions in the CS, the program can test any one of the Loopcounters. This is far less time consuming than scanning in one long program which tests all of the Loopcounters in one execution.

The Loopcounters test is the last of the control core tests. From this point on, the elements of the control core may be used to provide inputs to, and capture outputs from, the rest of the circuit modules. In particular, it is now possible to load programs into the CS to exercise the other modules.

This is known as boot-strapping, and it is widely used in the remainder of the tests.

Furthermore the technique, illustrated in the Loopcounters test, of using external flagging in association with an infinite loop to signal the end of execution of a sequence of instructions, is widely used in the following tests, together with the practise of only partially rewriting program segments. Combining these two results in a massive time saving, as many test sequences are repetitive in form, and thus only the changed instructions need be scanned in.

4.5 The ALU and Data Memory Decoders.

In considering tests for these decoders, note that, as discussed previously and illustrated in Figure 3.7, the inputs to these decoders is the IR, while the outputs are simple latches. By adding slave latches to each of the output latches and joining the master-slave latch pairs into a scan path, we can make the decoder outputs observable. The scan paths formed from the Decoder outputs in this fashion are tested for SA and API faults as part of the control core test discussed above. The IR may be controlled either by scanning or by down-loading instructions from the Control Store. If the latter method is used, as a single step, then a single instruction passes from the CS to the IR, then through the decoders to the data handling elements, as a consequence of each Single Step Strobe signal.

Each ALU field is 5 bits wide, and hence $2^5 = 32$ instructions are required to exhaustively test the ALU decoders. Their independence under

an OP instruction allows parallel testing of all four decoders. Each Data Memory field is 4 bits wide, and is independent of all other fields under both OP and LI instructions. If 32 unconditional OP instructions are constructed such that the ALU fields in any one instruction are identical, each instruction has unique ALU fields, all Data Memory fields in any given instruction are identical and each possible 4 bit binary pattern is represented at least once in the Data Memory fields of the set of 32 instructions, then the execution of that set of instructions constitutes the application of an exhaustive test set to all the ALU and Data Memory decoders in parallel. If the CS is preloaded with this test set, and the Single Step mode is used, then the test set can be applied one at a time, and the result scanned out of the decoder output latches for comparison off-chip after each test pattern.

As the Data Memory Decoders are invoked identically by both LI and OP instructions, they are independent of the LI/OP microcode bit. However the ALU Decoder fields are not independent of the LI/OP bit, as they are only valid for OP instructions, the microcode in these fields being interpreted as the LI data word under LI instructions. Furthermore, in each ALU the output latches which hold the Accumulator High Byte Load line and Low Byte Load line may also be set by a LI command to the Accumulator halves. Thus it is necessary to verify that under LI instructions, the ALU Decoder outputs are all null, except for the cases where the LI Destination is one or both of the Accumulator halves. (Null here is taken to mean that set of control line settings which produces no action in the data handling element — a local NOP.) This may be done using a series of unconditional LI instructions, with the LI data word chosen such that

those bits corresponding to the ALU fields would produce non-null results at all the decoder outputs, and the LI Destination field is cycled through its 32 possible bit patterns. The LI Destinations will include the Accumulator High and Low Byte Load, and this latter pair should be selected only after other patterns have verified that a LI command imposes a null output on the decoder output latches, over-riding the decoder outputs in LI instructions. This latter sequence also partially tests the interconnects from the LI Destination decoder to the various data handling elements.

For the sake of completeness, the bit positions corresponding to the Load Immediate Destination field of the OP instruction test set should be set to that pattern corresponding to the LI Destination BALAC — Both bytes, ALI ACcumulators. This will verify that the LI Destination is ignored during OP instructions. The content of the unspecified microcode fields in these instruction test sets is not critical, as under unconditional instructions the program can not branch, and the rest of the fields have no interaction with the ALU, Data Memory and Load Immediate Destination and Data fields, and hence any activity that occurs in the data handling elements or the PSR as a result of the test set may be ignored. The only proviso here is that some of the bit patterns applied to the ALU or to the Input and Output Processor (which share microcode space with the LI Destination field) may start an asynchronous process which may still be running or awaiting external input when the tests are complete *e.g.* multiplication, division or waiting for a Data Strobe In.

4.6 The Output Processor.

A few design decisions have been made to enhance the testability of the Output Processor (Figure 3.6). One is to locate the inputs of the Processor's data and control registers on the same segment of the Ringbus as the outputs of the Load Immediate bus-drivers (Figure 3.2). Another is to include two states in the LID Decoder, each of which activates a control line to latch one of the Output Processor's data registers: the Output Register 0 Latch line and the Output Register 1 Latch line, respectively. These measures allow the test to be made independent of the state of the Ringbus and switches, and removes the dependence on untested bus registers to provide the test data to exercise the Processor.

The only truly asynchronous feature of the Output Processor is the input OREQ, which may be set at any time by the external device. The rest of the Processor, from the clock-driven latch for OREQ inwards, is purely a synchronous device. As the prime concern in this test is the logical correctness of the implementation, the tests here are not concerned with the timing limitations of the OREQ signal: it is required to take the ideal value as per the handshaking specification.

The only difference between synchronous and asynchronous modes of operation are in the handling of the flag "DATA SENT". The flag is cleared, irrespective of mode, by the LI I/O Control Word being latched. In the asynchronous mode the flag is reset on phase 3 of a cycle if data has been sent: the OREQ is assumed to be unconditional. In the synchronous modes, however, time constraints require input requests to be

issued before the instruction conditional expression can be evaluated, and to overcome the problem of losing information that this creates, a handshaking protocol has been developed, which aborts incorrectly requested data transfers. The sending end Output Processor aborts by simply not setting the "DATA SENT" flag, thus preventing the information from being overwritten from the Ringbus. Thus the flag setting operation is depends on the synchronous/asynchronous selection, and the time variation of the OREQ line. On the other hand the flag clearing depends neither on the synchronous/asynchronous selection nor on OREQ, but only on the register and port configuration. Hence it can be assumed that if the flag setting circuit works correctly for one configuration, it will also work for other configurations with the same timing mode. Conversely, if the flag clearing circuit works for a particular configuration in one timing mode, it will also work correctly for that configuration in the other timing mode. Furthermore, verifying a particular configuration's data path connectivity is only required for one timing mode, as the paths are independent of the timing. Thus this test specifically verifies the flag setting circuit for one synchronous and one asynchronous mode, and tests the clearing circuit and connectivity for all configurations in one timing mode only.

The first test section selects the mode that most comprehensively exercises the Processor control and data handling, the synchronous mode in which both registers are transferred as two bytes in succession, one register's contents through each half port. The mode and initial register data is preloaded using LI commands, with the register contents chosen to be part of a test set for SA and API faults. The latch timing of OREQ is checked

by setting OREQ high on the non-latching phases, and checking that no output occurs. The internal flagging is checked concurrently by executing a WAIT "DATA SENT" instruction, which will generate a NOP until OREQ is latched and the output sent correctly. The OREQ line is then set for phase 1 only, which will cause the output to be written from the registers, allowing data and Strobe Out checking, but this is interpreted as the "abort" condition, and so the flag should not reset, and the NOP will stay high. The OREQ line is then set high for both phase 1 and phase 2, and this signals a correct output request, so the same data should be output from the port, but this time the NOP line should fall on the subsequent cycle, indicating that a successful output has been flagged to the PSR. Finally, the OREQ line is set high again for phases 1 and 2, and the output examined. This should produce no result, as the "DATA SENT" flag should block any attempt to retransmit data that has been output validly. Next one register is written, and OREQ set: no output should result as two register loads are required to clear the "DATA SENT" flag. The second register is loaded and the OREQ line is sent high simultaneously, and the resultant output (another of the API test set) verifies the "write-through" capability and part of the flag clearing circuit. The second register is reloaded with a new member of the API test set, and OREQ set high, but with no result. The first register is then reloaded, and OREQ initiates a successful output, completing the flag circuit clear checks for this configuration and set checks for the synchronous timing mode. The last of the API test set is loaded to the two registers and output, completing the API fault test of the registers, LI bus drivers, and all utilised data paths. Then two loads are done to

clear the "DATA SENT" flag, with the next instruction, a LI I/O Control Mode 0 without the Output Field Enable bit set, having its inability to set the flag verified. The result is flagged externally on the Strobe Out line by the concurrent OREQ, which will cause an output if the flag has not been set. The sequence is repeated with the Enable bit set to verify that the instruction can set the flag.

The test sequence thus far has verified the flag clear and connectivity for the configuration of mode 6, verified the flag set for synchronous operations, and checked all data paths for API and SA faults. There remains the verification of the flag set for asynchronous operations, and flag clear and connectivity checks for all other configurations. Note that the two untested synchronous configurations have asynchronous analogues, and that there are also three further untested asynchronous configurations. The next section of the test checks the connectivity, flag clear and flag set behaviour of asynchronous mode 0, and upon the completion of this test portion the remaining asynchronous modes are tested for flag clear and connectivity.

This completes the testing of the Output Processor apart from one small detail. The tests have used LI instructions to activate the data register latch lines, but in normal Op instructions these lines are driven directly by microcode bits in the Output Processor field of the instruction words. To verify this operation requires a source of data on the Ringbus which is accessible during Op instructions, but at this stage none has been verified. Thus the Output Processor is utilised in the subsequent tests of the elements communicating on the Ringbus, and correct results serve to verify the latch lines. If an incorrect result occurs before the operation of the latch lines

can be verified, then other data sources must be utilised to locate the fault to either the first data source, or to the latch lines.

4.7 The Ring Bus.

The Ringbus consists of sections of parallel metal buses connected into a ring by arrays of switches referred to as Pass/Break switches, and arrays of two-to-one multiplex elements with some additional logic, referred to as T-switches. The faults of interest for this type of structure are SA faults, arising from shorts to adjacent power rails, API faults from bridging both in the metal and in the active areas, and broken lines.

This test is designed to be carried out after the verification of the control core, thus allowing Program Execution to be used to apply instruction sequences, and it also assumes the pre-verification of the Output Processor and therefore of the LI bus drivers and the section of Ringbus common to these last two.

Note that the Ringbus is expected to have considerable capacitance, and thus to simply write to a register and then to immediately read the value back to the origin, does not necessarily preclude an intervening break in the bus lines: the capacitance of the section of broken line attached to the origin may retain adequate charge to write the checking register to the expected value. In these tests this is avoided by writing to the remote node and latching, then writing the bus to the opposite value, and then reading back the remote node to the checking node. The isolation capability of the switches is tested in a similar manner. This method requires the use of a

register on a bus segment remote from the LI bus drivers, which apply the initial patterns to the bus, and from the Output Processor registers, which capture the test response for relaying off-chip to the test assessor. Initially the data buffer of a DM pointer is used, and this requires verification prior to usage, but this test also verifies the Output Processor latch lines (as discussed in Section 4.6) and the interposed Pass/Break switch pass mode. Having verified that this register is settable via the switch, it is set, the bus reverse-charged, and then the switch opened, the register is read out and the Output Processor latched to test the isolation.

Having verified the isolation of the two bus segments, the Ringbus continuity can be tested by writing the API test set to the register, and then opening the switch and trying to pass the words right around the ring to the Output Processor, via both the upper and lower bytes of the 32 bit ALU bus segments (on different test cycles).

To test the T-switches' sign-extend and zero-fill capabilities, a register on the ALU bus must capture the generated byte, and output it later. The two T-switches at either end of the 32 bit bus cannot simply pass each other's generated bytes except in the all zero case, because this would cause bus contention. To this end the ALU Accumulators are partially verified for their ability to latch and output all zero and all one patterns. The full API test set is not used in this case because we have previously verified the bus segments' ability to pass this test set, and the T-switch generated bytes are only all zero or all one patterns.

Following this the Accumulators are used to hold the patterns required to test the isolation capabilities of the rest of the Pass/Break and T-

switches. They are used rather than the DM buffers because repeated reading of the DM buffers causes them to be re-written from the memory, and this operation has not yet been verified. This completes the Ringbus test.

The test consists of 135 executable instructions, and due to the unstructured nature of the test no looping is feasible. Thus the test is split into two independent blocks, and a pair of flagging infinite looping instructions are appended to each block, then the first is loaded and executed, and when the final flagging loop is detected the second block is loaded and executed.

4.8 The Input Processor.

The Input Processor (Figures 3.6, 3.2) embodies a significant amount of circuits which may operate in an asynchronous fashion, clocked by external signals through the Strobe In line. As the tests here are primarily concerned with verifying the logical correctness of the design and implementation, the clocking of the asynchronous portion of the test will be strictly controlled. As it is imperative that the Processor should be able to operate at least at synchronous speed, the asynchronous clocking will be run at that order of speed.

The register and port configurations are the same as for the Output Processor, while the flag operations are somewhat simpler. In the synchronous mode, the flag "DATA VALID" has no relevance, as the data transfers are always completed by a time which is locked to the program execution. The asynchronous mode sets the flag when the correct number of bytes (depen-

dent on the configuration selected) have been strobed in by the external data source.

The Output Processor, the Ringbus, and the control core are assumed to be previously verified. This allows the test to be run under Program Execution. The Output Processor is initialised to carry out two byte single register transfers: mode 7 has been arbitrarily selected. The Input Processor is initially set to mode 6: the synchronous mode in which all registers and all ports are exercised each transfer. This facilitates the testing of the registers, pins and paths for API and SA faults. A loop is executed by the CS program, initiating four inputs. An API test set which also flags the port to register connectivity is used to supply the data words at the pins for those four input operations. The words are transferred through to the Output Processor, which channels them off-chip. Before the first data word is read from the register to the bus, the bus is charged to the logical complement of the value expected from the register, in order to verify the bus driver capability. The second word read is the complement of the first, and after this various other juxtapositions occur, resulting in each driver being required at some time to drive the bus line to which it outputs from rail to rail in both directions. After this some tests are made to ensure that a failed conditional evaluation aborts an input request correctly, that the registers do not output to the Ringbus when they are not read and that the final two register write to bus selects correctly. This completes the flagging and data path fault testing for the synchronous mode, and two simple tests of connectivity for the other two synchronous modes completes this portion of the test.

Switching to the asynchronous mode 1 (single byte), tests are performed to ensure that no latching occurs in the absence of a Strobe In pulse, the connectivity is correct, the mode is only single byte and the flag is set on completion. A failing TEST conditional evaluation is used to verify that the IREQ line is not set for failed conditional input initiates, and a pair of timing tests are used to verify the flag to PSR timing. This completes the general asynchronous testing, as well as the flag testing and the connectivity test for this particular single byte asynchronous mode. The other single byte asynchronous mode 3 is tested next for connectivity and flagging, and in a similar fashion the parallel double byte mode 0 is tested.

The last two modes, the dual sequential byte modes 2 and 4, are tested last, using similar tests. First one byte is strobed in, and the registers read to ensure correct connectivity, then the flag is examined and the program idles until the second strobe in occurs, the subsequent outputs confirming the connectivity for the mode. A final strobe pulse is added to attempt to corrupt the registers with an extra load, and the registers are output again to verify the failure of this event.

This completes the testing for logical faults, and because the asynchronous clock latching periods have been the same as the synchronous latching periods, we know that the Processor will function within a reasonable range of clocking speeds. How far that range can be extended towards longer or shorter Strobe pulses will require definition, but it would seem to be a rather ill-defined multi-variable analogue testing problem, with the three system clocks, the IREQ to Strobe delay and the Strobe pulse width giving plenty of scope for variation, and requiring some reasonably sophis-

ticated equipment. The method would probably be to Strobe in data with alternating bit patterns, varying one delay, period or pulse width at a time until the data patterns are no longer transferred accurately.

4.9 The Data Memories.

The Data Memories' only outputs are onto the Ring Bus, while their inputs come via the Ring Bus, the NOP line and the Data Memory Decoders' outputs (Figures 3.2, 3.3). If the Ring Bus is controllable and observable, and the NOP line and Decoder outputs controllable, then each Data Memory may be tested as follows. The test sequence in Appendix A is detailed in terms of instructions to be executed under Program Execution, but in the case of an Execution Controller or Decoder failure, the bit patterns may simply be scanned in, and applied by the manipulation of system clocks. The writes and LIs to memory are generally carried out on all four memories in parallel, with the outputs being passed to the pins, one memory at a time, via the Ring Bus and the Output Processor.

First the data buffer registers associated with each memory pointer are tested for Stuck-at (SA) and Adjacent Pattern Interference (API) faults, and for their ability to latch, hold and output bits to and from the data bus. This is done by writing, and immediately thereafter reading, data to and from the buffer under test: the immediacy is required to prevent the buffer being over-written from the memory array itself, which would occur if the words were written to memory in a block, and then read back. The data written to the buffers is chosen to detect the set of API faults under

consideration, single static pattern dependence to the nearest neighbour cell or data line, as well as SA faults.

The columns, as in the Control Store, are assumed to have no API faults, due to the isolating effects of the intercolumn voltage rails, and are thus tested simply for SA faults, by writing ones and zeros to each column, and reading them back. The row selected for this test is not critical. Further, as the row may contain cell faults, a suspected column fault should be confirmed by retesting the column at another (distant) row.

Cell faults are postulated to be of the same form as those in the CS array, and thus a similar form of test is used to verify the DM cells *i.e.* a variant of the MARCH test. Direct addressing is effected by loading the pointers each time a write or read is required: somewhat time consuming, but the relative addressing circuits cannot be tested until the absolute addressing and cell tests are complete. Similarly to the CS test, both pointers are used, and the results of this test provide proof of the uniqueness of the decoded wordlines, and hence of the addresses, for both pointers. The CS test explicitly tests this feature, because precise fault location information is required for reconfiguration, whereas in this case an inferred pass or fail result suffices, as no reconfiguration is possible. A further test checks for correlation of addresses and wordlines between the two pointers, by loading each address with an unique word using one pointer, and reading it back from the same address using the other pointer. The words left in the memory are used in the subsequent tests as flags to indicate which address has been generated by the automatic pointer update circuits.

To this point, the tests have comprised of groups of instructions to be

loaded and executed in a non-looping manner, with efforts being made to minimise the number of instructions required to complete a test, thus economising on test scan-in and execute time. The pointers' auto-update circuits each contain an adder, which may either increment by one, or add a preset increment to the current address. Adders may be tested pseudo-exhaustively (as per [82, page 15]), but in the increment by one case this results in a slow test, because each of the specific test states has to be scanned in to load the CS, and then executed. The strategy actually employed is to load instructions to read each pointer with an increment of one and output the resultant flag word, embedded in a loop which executes thirty two times. This is effectively an exhaustive test of the increment by one mode of the pointers' operation. The increment by one test covers about half of the possible states of the adders, with the other half only being exercised in the preset increment operations. Here it is impractical to cover all the possible states exhaustively, so a minimum size pseudo-exhaustive test is implemented. This completes the test of the pointer control adders, and of the pointer control circuits *in toto*.

4.10 The Adder/Subtractor/Shifter/ Accumulators.

The Adder/Subtractor/Shifter/Accumulators (ASSAs) are tested as functional units by executing a test program stored in the CS, which applies normal instructions, together with selected test data (Figures 3.2, 3.5). It

is assumed that the control core, the ALU Decoders, the Ringbus and the Output Processor have been previously verified. As a matter of convenience it is also assumed that the Data Memories have been verified, so that the API test set may be stored there, rather than using an LI to generate the patterns each time. This is not critical to the test in any way: the LI method is just slower.

The ASSA consists of several distinct circuit partitions physically: the shifter/accumulator is one, the adder is another, and the two input stages are also distinct. The ALU instruction set has a sufficiently broad scope of operation combinations to allow most of these partitions to be tested either independently or in a "bootstrapped" fashion, using the previously verified results for one partition to apply tests to or capture results from another partition.

The instruction set includes operations involving only the shifter/accumulator register, and these are used as a starting point for the test, as all the other partitions' output is channelled via the accumulator. Although the Ringbus test does actually include some tests on the accumulator's latch, hold and output capabilities, these are ignored in this test, to make this test a stand-alone item which can be applied, for instance, to the single multiplier to be fabricated as a part of the design verification process. The test sequence verifies the shifter/accumulator's ability to latch the static API and SA test set, output it to the bus, and shift it in either direction, and also tests the accumulator's ability to generate the "ACC < 0" flag from its most significant bit and to propagate it to the PSR.

For the sake of compactness the tests are intermingled in a manner

that utilises, as far as possible, the result of one operation as the input for the next test. These latch, shift and drive operations do not affect or involve the adder or input stages at all. Each test instruction is applied to all shifter/accumulators in parallel, with the results being driven off-chip, one partition at a time, via the Ringbus and the Output Processor. This section of the test concludes with the CLear ACCumulator (CLRACC) instruction, which is actually a precursor to the second half of the test, because it tests that the Accumulator latches the all zero output from the adder, that each input stage can apply the all zero pattern, and that the adder cells add correctly the $0 + 0 + 0$ pattern. It is undesirable, but also unavoidable, that this single test verifies four different operations, one associated with each partition. If a fault does occur, however, it is possible to utilise further instructions (not detailed here) to locate the fault to one or other of the partitions.

The second section of the test consists of four closed loops, in each of which a single one is loaded into the accumulator, and is shifted across until it shifts out of the most significant end of the shifter. A test is applied once each loop execution to check the flag "ACC= 0", and it should fail until the single bit shifts out, thus setting the NOP line externally and inferring the correct operation of the flag generation circuit. Upon detecting the zero condition, the TEST instruction enables a jump out to the next loop, which performs an identical test on the next accumulator.

In the subsequent sections of the test, patterns are applied to the adder via the bus and accumulator input stages, and captured by the accumulator for output via the Ringbus and Output Processor. The patterns applied

include an API and SA test set for both the inverting and non-inverting lines of each input stage, and for the output of the adder and connection to the accumulator, and also include the pseudo-exhaustive or path-sensitised test set for the adder circuit: there is some overlap between these test sets. Wherever possible, each test builds on prior results in such a way that the number of new conditions tested is minimised. This increases the fault location capability of the test, albeit at the cost of a longer test sequence, since a single instruction which verifies multiple conditions may replace several tests for the specific conditions, but a failure in this multiple testing case will not necessarily be locatable.

In the third section of the test, the patterns applied are all zero or all one, and are generated by using the T-switches to expand the 16 bit patterns obtained from the DM to 32 bit patterns on the ALU bus. As in the first section, the lack of regularity or repetition in the test precludes the use of program looping to reduce the number of instructions, and hence loading time, required.

In the fourth section, alternating one and zero 32 bit patterns are required, and these are created by writing a 16 bit alternating pattern to the high byte and the low byte of the accumulators in succession, thereby assembling the 32 bit pattern. As these two patterns are both applied in identical test sequences, the test is performed as a loop, with the data memory pointers' automatic updating being used to fetch the appropriate data in each loop. This section completes the verification of the input stages, and all subsequent tests are to complete the adder verification.

The fifth and final section includes the final test for the individual bit

adder cells, and the final eight tests required to complete verification of the carry lookahead cells. The carry lookahead tests all require specific and unrelated patterns to be applied, so a loop is used to execute a generic format test eight times with different data fetched from the Input Processor each time.

4.11 The Multiplier/Dividers.

In its operational form, the multiplier/divider (Figures 3.2, 3.4) provides a difficult testing task. The control circuit causes iterative adder operations to occur between input and output, the algorithms call for data-dependent scaling operations on the adder inputs, there is no direct path between the inputs and the output, and there is no way of examining internal intermediate results. Taken together, these features ensure that logical partitioning and path sensitisation are incapable of testing the module in its operational configuration. The exhaustive test requires 2^{32} operations for the multiplier alone and is obviously out of the question, while a reduced functional test set offers no guarantee of fault detection and no indication of fault coverage.

The solution is not to attempt to test it in operational form, but to reduce the problem to manageable proportions. While this may be possible using scan paths to control and observe registers, an initial consideration of this approach highlighted a major problem: the multiple or iterative adder operations, if left unchecked or uncontrolled, results in the overlaying and confusion of many test responses, so that an observed faulty response may not be directly attributable to any particular section of the module. To

overcome this requires that the control circuit be augmented to allow preset stops or starts.

The test strategy finally implemented derives from these considerations and, simply stated, the idea is to stop the module after only one adder operation, and cause the interim result to appear at the output register. This simplifies the observability and controllability problems immensely, and allows testing by sensitising paths through the module from the inputs to the output. This in turn allows the test to address portions of the circuit while holding the others in a known state or in a previously verified mode of operation. The method is equally applicable in multiplication and in division, and in practice is required in both cases to fully test the module.

The details of the implementation of this test state are detailed later in Section 4.12.2. Briefly, a new control bit is added, driven by the ALU decoders upon selection of one of a number of new test operations in the ALU instruction set. These test operations set the same control lines as their operational equivalents, and also set the TEST line. The TEST line is latched in at the same time as the GO line and the M/D line, and while the multiplier/divider commences its first operation cycle, the TEST bit is fed to the circuit that terminates the module's operation and organises the write to the output buffer.

As the multiply operation clears its internal accumulator upon commencement of the multiplication, the first adder operation simply adds the value in the parallel input register D, scaled by 0 or 1, to the all zero accumulator, or it may subtract the value, scaled by 1 or 2, by applying the logical complement to the adder input while also adding a carry-in at the

least significant bit of the adder. Clearly this is a sensitised path from the input D to the output. The scaling and operation sign is controlled by the values in the two least significant bit positions of the Delocalised Multiplier Input (DMI). If the DMI value is chosen to give a scaling of +1, the output will track the input D. Applying the API test set at D will verify D, and the path from D through the adder cells to and including the output buffer. Repeating this for the other relevant values of the DMI least significant bit pair verifies most of the input stage logic and most of the adder cell operations with the accumulator set to zero. The sign extend on right shifting (at the adder output) is verified, as is the right shift in the high byte. These tests are implemented by nesting loops of instructions so that, for each of four DMI values, the program successively applies the four API test patterns to the D inputs of the multipliers (operating in parallel), and executes TEST multiplication instructions, after which the results are captured from the Ringbus by the Output Processor and exported off-chip.

The next section of test once again uses a loop to apply a succession of TEST division instructions. In division, the accumulator is loaded with the dividend at the beginning of the operation, and the contents of the D register are added or subtracted from the dividend according to the result of the exclusive nor of the most significant bit of each of the two numbers. This EXNOR result is latched into the least significant bit of the accumulator/shifter chain, giving an observable output to the exnor cell. Using the dividend load in conjunction with the previously verified input stage, the division load port may be verified for the API test set, and then the adder verified for non-zero inputs from the accumulator. Most of

the carry lookahead circuits are also verified at this stage, as is the shift left capability. The two highest bit adder cells and the most significant carry lookahead cell are not verifiable under this form of instruction, as the left shift from the adder output into the accumulator renders the outputs from these two adder cells unobservable. In the operational multiplication mode, these bits shift right, and by choosing the DMI value appropriately it is possible to deduce from the result at the output buffer if those adder cells worked correctly.

The third section of the test is a piece of straight-line coding which tests that the modules do not start if the global NOP is high, that multiplication takes nine clock cycles, and that the flag "MD n COMPLETE" is set at the end of the operation. No data is written to or read from the multipliers in this test, as this would require loading multiple copies of the I/O instructions, and the saving in execution time from reducing the test set for the next section of test is far outweighed by the additional scan in time for instruction loads. Instead the NOP external pin can be used to monitor the progress of the execution, along with the IREQ line, which is set as an external flag by the Input Initiate bits in the instructions.

As inferred above, the fourth section of the test is a loop executing a number of full multiplications. The data set for these multiplications is chosen to verify the load and shift capabilities of the DMI, to complete the testing of the input selectors (scaling by +2 and by -0 cannot be achieved in the TEST multiplication), and to test the remaining unverified bit adder and carry lookahead cells. The data is loaded from the pins in response to the IREQ input request.

The fifth section of the test is identical in form and function to the third section, with the one difference being that the instruction under test is division. The sixth and final block of the test is a loop that executes full divisions on a data set supplied from the pins via the Input Processor. These are required to verify the operation of the overflow detection and hard-limiting circuits.

4.12 The Built-In Test Hardware.

As has been mentioned in the foregoing sections, there have been some specific design constraints on, and additions and alterations to the TFB circuit to enhance the testability of the chip *in toto*, and of particular partitions.

The most systematic of these measures is the insistence on the use of pseudo-static latches for data capture, rather than using charge storage on dynamic nodes. Further to this, efforts have been made to minimise the amount of dynamic or domino logic implemented, and the only major examples on chip are the decoders for the CS pointers IPC and JPC, where the required speed of operation precludes the use of static logic. These measures, while not directly conferring observability or controllability to any element of the circuit, ensure that the chip's operation can be suspended while the scan paths are used, without the penalty of having to reinstate the former values prior to restart.

A simple measure which considerably eases the problem of observability of the Execution Controller is to output the global NOP line via an oth-

erwise spare pin. This output is heavily utilised to determine the results of instructions which test the conditional evaluation circuits, and when these circuits have been verified it is used to accelerate the detection and verification of flag returns from the data handling modules.

The two major design initiatives for the enhancement of chip testability are the additions and alterations to the multiplier/dividers' control circuits to allow test operations of a non-functional nature, and the inclusion of a system of scan paths through the control core to facilitate certain functional requirements of the chip while also allowing test application and observation. Each of these is discussed in more detail in the following sections.

The only other major hardware overhead for testability is the I/O porting. Perusal of the TFB pinout, tabulated in Table 4.1, shows that of a total of 63 pins allocated, 50 of those are allocated for purely operational purposes, 8 pins are utilised for both functional and test purposes, while only 5 pins may be considered to be solely allocated for testing purposes. The fact that 63 pins are allocated is not sheerly fortuitous: the functional requirement for over 50 pins means that the smallest size standard package that could be used is a 64 pin dual in-line package. If the pin count was allowed to expand beyond 64, the packaging would have to use a pin-grid array, which would add significantly to the packaging costs, and so the test hardware has been tailored to keep within this pin limit. Leaving a pin unallocated is also part of the design philosophy of having some reserve capability: if later in the development cycle a pressing need arises for extra observability, controllability or functionality, this pin may be utilised without requiring a major redesign.

| Port Name or Function | No. of Pins | Operational or Test |
|------------------------------|-------------|---------------------|
| V_{DD} Voltage Rail | 2 | Operational |
| V_{SS} Voltage Rail | 2 | Operational |
| Clock Phase 1 | 2 | Operational |
| Clock Phase 2 | 2 | Operational |
| Clock Phase 3 | 2 | Operational |
| Data Input Port | 16 | Operational |
| IREQ | 1 | Operational |
| Strobe In | 1 | Operational |
| Data Output Port | 16 | Operational |
| OREQ | 1 | Operational |
| Strobe Out | 1 | Operational |
| Single Step/Program Execute | 1 | Operational |
| Single Step Strobe | 1 | Operational |
| Write | 1 | Operational |
| Control Scan 1 In | 1 | Both |
| Control Scan 1 Out | 1 | Both |
| Control Scan 2 In | 1 | Both |
| Control Scan 2 Out | 1 | Both |
| Scan Clock 1 | 1 | Both |
| Scan Clock 2 | 1 | Both |
| Decoder/Control Scan Select | 1 | Both |
| Upper/Lower Scan Select | 1 | Both |
| ALU/DM Decoders A,B Scan In | 1 | Test |
| ALU/DM Decoders A,B Scan Out | 1 | Test |
| ALU/DM Decoders C,D Scan In | 1 | Test |
| ALU/DM Decoders C,D Scan Out | 1 | Test |
| NOP Output | 1 | Test |

Table 4.1: The TFB Pin Allocations.

4.12.1 The Scan Paths.

As was outlined in earlier sections on test strategy and the control core test, the functional requirements of the chip predispose the design to the introduction of serial input paths to load program instructions, to initialise the BARs, to initialise the column switching and to set some control bits that are used for pointer selection in Write operations. On top of this, the architecture is not conducive to parallel access to the memory, and yet an efficient and effective test of memory is required, to provide information on the row, column and cell availability, adequately detailed to facilitate electronic sparing. This last requirement is met by the addition of the wordline latches, joined together to form a scan path, together with the conversion of the IR into a scan path segment. It was also noted that the data set for the CS cell test could be generated simply by clearing or setting the Write Register, rather than by scanning the full data word in, and provision was made for two bits to be scanned in with the other Control Input bits to initiate either a clear or a set in the register, as required.

The testing of the control core introduces additional requirements for the inclusion of registers in scan paths. It was found that the inclusion of the pointers IPC and JPC, the flag register PSR, and the outputs of the LID, ALU and DM Decoders, as elements of a scan path, was essential to the efficient testing of this partition.

Taking together the functional requirements for serial data inputs, the requirements for facilitating the CS test to allow reconfiguration, and the necessities of access for testing, a substantial number of elements must be

assembled into one or more scannable paths. The best way to configure this path, or paths, depends on several criteria. Consideration must be taken of the effect of the configuration on yield, on the ease of use, on the implications for the test equipment. Note that the configuration should take into account the physical layout or floorplan of the chip with regards as to which elements are required to be connected together, and in what order they are connected.

Though it can be shown by analysis (see Appendix C) that one single scan path incorporating all the required scan path elements has a lower probability of sustaining a fault than multiple scan paths sharing the elements between them, the same analysis shows that for the real cases under consideration here, where the scan path elements far outnumber the multiplexing elements, the decrease in absolute yield due to adding the multiplexing elements is marginal, while a substantial advantage accrues through the increased probability of getting some data from the paths. The maximum benefit from splitting the scan into a given number of paths is gained when all branches are identical: this is not strictly feasible when the physical grouping of various scan path elements is considered, but the path lengths should be kept of roughly the same order of magnitude wherever groupings and routing allow.

In terms of convenience, or ease of use, the multiple paths are preferable to a single path from several viewpoints. First, the longer the path, the longer the time to scan information into and out of it, and this is particularly inefficient when in any given test only a portion of the data is of interest, but the scan must be cycled completely to restore critical val-

ues. Also, having a long scan path imposes a heavier demand on the test hardware required to feed this serial string into the chip. More memory is required to store the longer strings, and the test controller must be able to hold the other inputs constant for longer while applying this serial input from a much larger parallel to serial conversion unit. The same criticisms also apply to program loading in normal operations. Furthermore, when the multiple paths use separate I/O pins, and are capable of being shifted in parallel, the data transfer time is halved, allowing an enormous speed-up in test and program load procedures.

Given the above considerations, and the availability of up to fourteen pins for testing purposes, the following system was designed (Figure 4.1). Four scan paths are embedded in the control core, using control elements suitably modified and connected. These four paths, of similar but unequal lengths, are arranged as two pairs, with each pair being multiplexed together at the input pin and at the output pin. There are a further four scan paths, one through the output latches of the ALU and DM Decoders in each quadrant of the chip. These too are organised as pairs of paths multiplexed together at the inputs and outputs.

It was considered desirable to retain the ability to shift each path independently of the others, and yet it is highly desirable to be able to access certain pairs of paths in parallel. For instance, the Control Inputs path must be loaded prior to writing to the CS, as must the Write Register if a specific instruction is to be loaded, so these two elements are placed on separate paths which are clockable in parallel. The path selection and clocking is designed to facilitate these aims. There are two independent scan clocks,

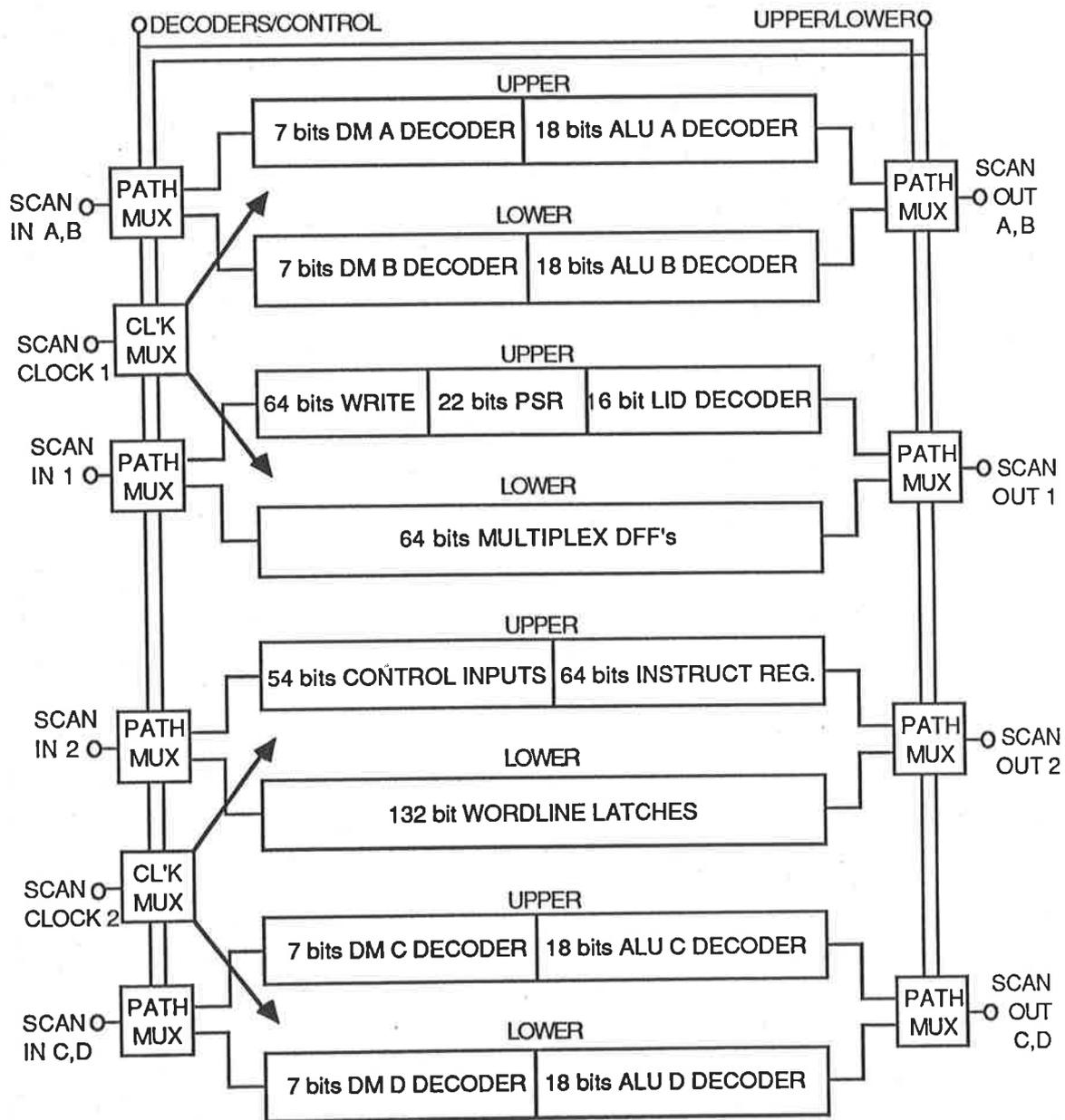


Figure 4.1: The Scan Path System.

each connected uniquely to one pair of multiplexed control scan paths, and to one pair of multiplexed decoder scan paths. Both clocks, when activated, drive the paths selected by the two Scan Select lines, Decoder/Control and Upper/Lower. By activating a single clock only, any one scan path may be individually cycled, while activating both clocks simultaneously scans out predetermined pairs of paths, on separate pins. The organisation of paths into multiplexed pairs and clock sharing pairs is detailed in Table 4.2, as is information regarding which elements constitute which path, and in what order they are arranged.

The arrangement of the control scan path elements into complete paths can be argued as follows. The Control Inputs should be arranged in parallel with the Write Register to allow an efficient Write to the CS, while the IR and the LID Decoder need to be observed during the control core test, and so should be placed near the output pins. The PSR needs to be loadable but this facility is not used frequently, and so the PSR can be placed further from the inputs than the Control Inputs or the Write Register. The wordline latches, as the biggest homogeneous group, are kept together. It would reduce test times in some sections if the wordline latches were scanned in parallel to the Control Inputs, rather than the WR, but overall the test time would suffer a significant increase. The IR needs to be read during the same test procedures that the CI and WR loads take place in, so it makes sense to concatenate two of these segments into one path. The remaining segment is connected up with the PSR and LID Decoder to form a path, while the multiplexor DFF chain is left as a stand-alone path, since it is the least often changed.

| Scan Path Selection | | | Path Elements (top to bottom, input to output) | Path Reg's Count |
|------------------------|-----------------|---------------------------------|--|----------------------------------|
| Pins | Clock | Upper/Lower Decoders/Control | | |
| Control Scan 1 | Scan Clock 1 | Upper Control | Write Register PSR LID Decoder | 64 22 16 $\Sigma = 102$ |
| | | Lower Control | Multiplex DFFs | 64 |
| Control Scan 2 | Scan Clock 2 | Upper Control | Control Inputs Instruction Reg. | 54 64 $\Sigma = 118$ |
| | | Lower Control | Wordline Latches | 132 |
| Decoder Scan A,B | Scan Clock 1 | Upper Decoders | DM Decoder A ALU Decoder A | 7 18 $\Sigma = 25$ |
| | | Lower Decoders | DM Decoder B ALU Decoder B | 7 18 $\Sigma = 25$ |
| Decoder Scan C,D | Scan Clock 2 | Upper Decoders | DM Decoder C ALU Decoder C | 7 18 $\Sigma = 25$ |
| | | Lower Decoders | DM Decoder D ALU Decoder D | 7 18 $\Sigma = 25$ |

Table 4.2: Scan Path Organisation.

The multiplexing is arranged as follows: of each pair of multiplexed paths, one is connected to the pins via the multiplexors by the Upper/Lower line being held high, and the other is connected by that line being held low. A simple transmission gate multiplexor cell suffices here, as speed is not a problem, and there will be restoring logic on either side of the gates. This is illustrated in Figure 4.2. Each clock is associated uniquely with one Control Scan pair and with one Decoder Scan pair. Decoder/Control selects to which of these two pairs the clock is fed, and the Upper/Lower line is again utilised, this time to switch the clock to the appropriate path. This arrangement allows the selection of paths to be carried out by the distributed processing of logic levels, rather than by a centralised switching of various clocked lines. The clock switching circuit will require a substantial driver for each path's clock line, and when that path is not selected, the clock line must be held low. This requires a more complex switching and driving arrangement, as illustrated in Figure 4.3.

A modified master-slave scan path latch has been designed and is illustrated in Figure 4.4, and this may be compared to the standard master-slave latch illustrated in Figure 4.5. The actual implementation of the scan paths has not yet been carried out, as the control core layout is still awaiting completion, as are the the ALU and DM Decoder layouts, and consequently the final assembly and routing stages are not complete either. Hence the physical sizing, positioning and routing of the scan path elements are debatable at this stage, as are any estimates of line capacitances, so the sizing of any drivers or logic simulation of large scan paths is precluded until this information is available.

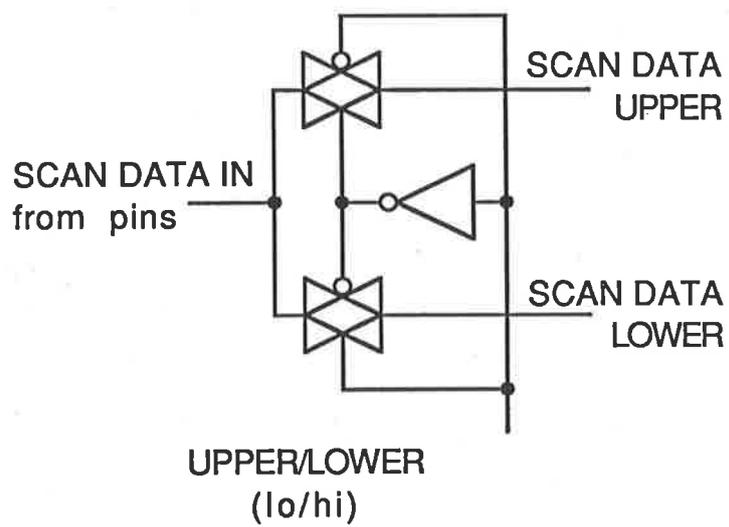


Figure 4.2: CMOS Transmission Gate Multiplexor.

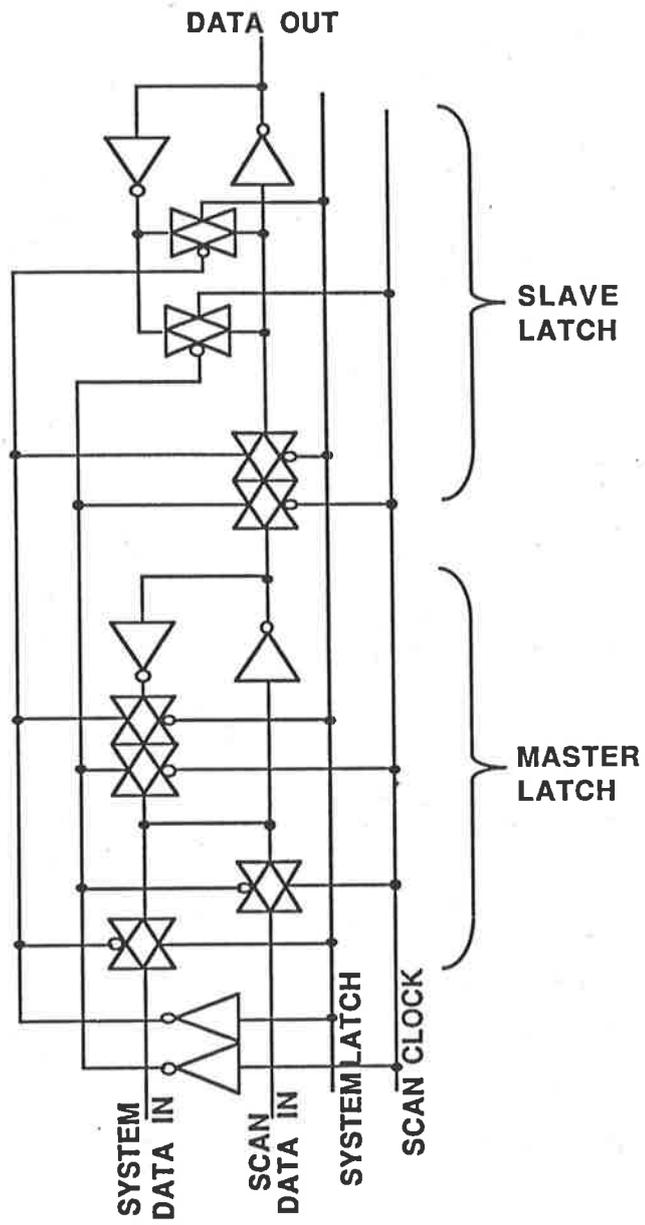


Figure 4.4: CMOS Scan Path Latch.

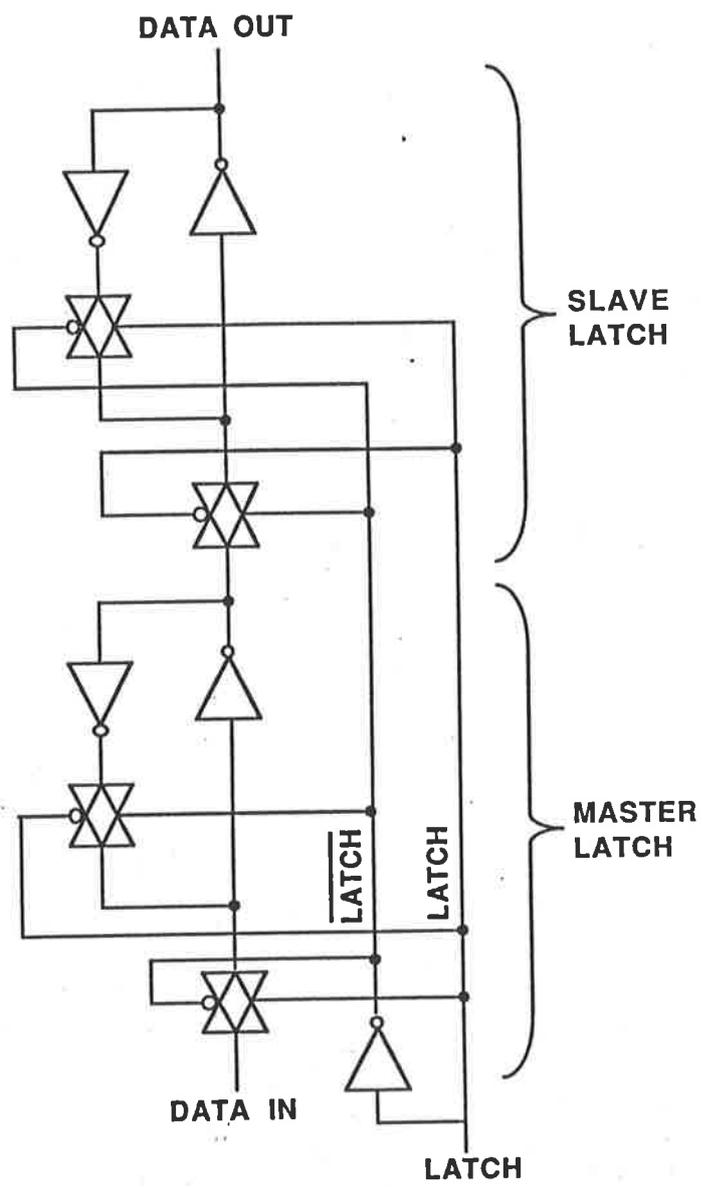


Figure 4.5: CMOS Master-Slave Latch.

4.12.2 The Multiplier/Divider Modifications.

There are two modifications to the multiplier/divider circuit to make the module testable. The major change is the addition of a TEST bit latch, latching on the same conditions as the GO and M/D bits of control. This latch is fed to the circuit that in normal operations accepts the time-out return from the control timer, and is processed in the same manner as the time-out return, thereby bringing the module to a halt after only one adder operation and filling the output buffer with the result of that operation, shifted appropriately. The actual circuit alterations consist of one new pseudo-static latch (master only, not a master-slave pair), and one gate which combines the time-out returns for division and multiplication is augmented by one further input to combine in the TEST signal in identical fashion. The ALU Decoder has one extra output line TEST added, and several states are added to the instruction set to produce the single cycle of multiplication or division, which results in some increase in the Decoder size. The exact overhead is not known, because the decoders, while being completely specified, have not been laid out as yet.

The minor change to the module concerns the overflow detection circuit. In the original design, the overflow circuit output was not valid until the start of the third clock cycle of operation. In fact the implementation forced the output of the circuit to flag overflow until the last of three control bits became available and was latched during the SECOND adder operation. This in turn caused the hard-limiting cells between the internal register and the output buffer to hard-limit until the second adder operation. This

renders the TEST operations useless, as all that is available at the output buffer is either plus or minus full range. However all that is necessary to change this is to disable the "OVERFLOW" output from the cell until some time after the end of the first adder operation. Fortunately, there is already a signal available within the the overflow circuit that makes a transition at the appropriate time; the third latching signal and its complement make transitions, from one to zero and vice versa respectively, in the second adder cycle. Now the overflow signal is already ANDed with the M/D line to ensure that hard-limiting only occurs during division. If the gate used for this is extended to become a three input gate, and the complement of the third latching signal is fed to that new input, then the overflow output is disabled until the second adder operation. This allows the data in the TEST instruction to be output without hard-limiting, while leaving the hard-limiting operation unaffected for full term operations. The overhead for this change is negligible: two transistors to convert a two input gate into a three input gate, and a short length of polysilicon to connect the new gate input to a signal already existing in the cell. The implementation of both this and the previously discussed major change has been assigned to the designer of the multiplier/divider unit, to be done concurrently with some other work required.

4.13 Summary of the Complete Chip Test.

To summarise the complete chip test is essentially to reiterate the preceding test sections in the order that they appear here. The control core is verified

first, piece by piece, starting with the scan paths and gradually expanding the core of verified modules. As more modules are verified, the tests become easier to apply, and the core operations become more like normal program execution. Once the core is verified, Single Step operations interleaved with scans are used to verify the instruction decoders. Upon the completion of this test the complete instruction application hierarchy has been verified. The next step is to use the currently verified hierarchy to test the data communication channel, starting with the Output Processor, moving next to the complete bus, and finally to the Input Processor. When these modules have been characterised, they become tools to verify the data processing elements. Here the parallel nature of the architecture really becomes apparent, in the way that the testing can be applied simultaneously to all similar modules. The testing of the data handling modules, and in general of all the modules or elements, relies on knowledge of the module structure to produce tests sets that are close to minimal. Exhaustive tests are only used where significant time advantages are evident, or there is insufficient structural data upon which a test can be based.

Chapter 5

Conclusion.

5.1 Test Methods.

Reviewing the test procedures, it can be seen that there are two different methods used to apply tests to the various partitions. Before considering these two separately, remember that there is commonality in the approaches: in both cases the same fault models are used, and consequently the types of patterns applied are the same. The difference lies only in the method of application of the tests.

5.1.1 Design for Test.

The first method, as used in the verification of the Control Core, requires the inclusion of specific test structures, the scan paths, at the initial design stages in order to facilitate the application of test patterns, and the capture and export of the test results.

The use of the scan path methodology is justifiable in terms of the minimal hardware overhead, the ease of application of the methodology, and most importantly the ease of integration of the test system into the functional architecture.

This last point is clearly illustrated by the fact that the normal chip functions of loading the instructions into the Control Store, and reconfiguring the RAM array to carry out row and column sparing, dictate the use of serial paths, due to the nature of the architecture and the lack of parallel access to the control core. These same serial paths are used to apply test patterns and to control the various elements of the control core during the test.

Another important aspect of the integration of system function with test requirements is the way in which the Single Step operation is organised, in so far as it facilitates the system function of software debugging, and at the same time is implemented in a manner that allows the safe use of the serial input and output paths through the chip without disabling the system clocks, thus also facilitating the testing of the chip.

The major penalties for adopting a scan path test system are the enormous overhead in scan time (compared to the application time of the patterns), the increase in routing complexity incurred by the necessity of connecting all the sequential elements, and the vulnerability of the test system to single point faults.

Addressing first the problem of routing complexity, this is ameliorated by several factors. First is the use of existing large parallel register structures as the base unit into which the scan path is introduced. These struc-

tures are laid out as iterations of identical bit-slice cells, with the control lines running in channels through the bit-slices in such a way that their abuttal connects the control line segments into single complete lines. This reduces the scan clock routing within the register structure to a minor problem, as it may simply implemented as another segment in the control line channel. The actual scan path connections, for passing the serial information from cell to cell, is handled in a similar fashion, although the routing channel is not used, but instead an input is placed at one edge of the bit-slice cell, while the output port is placed at the other edge, in a position pitch-matched to the input. Abuttal of the cells then serves to connect the scan path cells into a single serial path segment.

The connection problem then reduces to the problem of routing two lines from one major register block to the next included in the scan path, and in each case there are three or less of these blocks per path. This problem is also eased by the fact that all of the elements chosen to be connected into any single paths are in the same area of the chip, so long routing paths are generally not required, except maybe to connect out to the pin pads. It is further eased by the fact that the control core, through which the major paths connect, is not a highly structured region, and thus there is a reasonable amount of routing space between the various elements requiring connection.

Again, it may be argued that the system architecture allows no other simple implementation of the instruction loading or reconfiguration, and so the use of these facilities for testing incurs no extra penalty with regards to the routing or single point failure vulnerability. Efforts have been made

to reduce the system's single point failure vulnerability by splitting the scanned elements up between separate scan paths (see the discussion in Appendix C).

With regard to the test time overhead due to scan loading and unloading of patterns, there is no denying that this is significant. Perusal of the table of test lengths at the end of Appendix 1 shows that, averaged over the complete chip test, the scan times form around 95% of the total, with the actual application of the tests taking only 5% of the time. This is balanced however by the nature of the tests that the scan paths allow: often a more direct test of the particular elements is possible by the inclusion of scan paths, whereas their exclusion leads to the use of large functional test sets that arrive at the result inferentially after testing a large number of combinations of states.

This last point can best be illustrated by comparing the Control Store tests with those of the Data Memory tests. For the 8 kilobit Control Store, the scan path oriented testing takes some 117,248 clock cycles, while the Data Memory functionally driven test takes about 12,000 cycles (two fifths of the total DM test time) to test a 512 bit memory: the CS is 16 times as large as a single DM, and yet takes only 10 times the test length. (It is only fair to point out, however, that the DM test time is largely a function of the scan time anyway, as the tests must be serially loaded before execution.)

5.1.2 Functional Testing.

The second method, which is essentially a classic MSI approach, is to use the previously verified system components to apply the tests to the partition under test, and therefore the tests are limited to subsets of normal functional operations, together with subsets of the allowable data. As has been illustrated in the previous chapter, most of the data-handling modules of TFB are of such a form that they allow complete testing of the postulated fault set by this method without alteration.

The exception to this are the multiplier/divider modules, which require additional circuitry to make them controllable and observable within the framework of a functional test scheme. This additional circuitry, when activated, changes the modules' function from an iterative operation mode to a single operation mode, thus allowing single tests to be applied and checked. In fact these modules may well be testable using a purely functional test without the extra control circuitry, but the task of fault location is certainly difficult, if not impossible, in this case, and the task of generating the tests by hand is of similar complexity. The very minor area overhead incurred to implement this test circuit is more than amply compensated for by the increase in fault location it affords.

5.2 Testing Overheads.

A single bit master-slave latch requires 4 extra transmission gates and a clock inverter to convert it into a scan path latch, so this conversion gives rise to a 55% greater transistor count for the existing registers requiring

inclusion in the scan paths. Each new slave latch added to convert an existing master latch into a scan path element incurs an extra 6 transmission gates and 3 inverters, while each completely new scan path cell adds 8 transmission gates and 6 inverters.

The only registers and latches where the conversion to, or addition of scan path elements is purely a test overhead, are the PSR (22 bits), the IR (64 bits), the LID, DM and ALU Decoder output latches (22 bits, 4 x 7 bits, and 4 x 18 bits, respectively), two Control Input registers (2 x 1 bit), and the Wordline Latches (132 bits). Together these add about 6800 transistors to the circuit. Add to this the small amount of switching logic to select the paths, and the scan clock drivers, and the overhead comes to around 7000 transistors. The other overhead associated with the scan path system is the I/O for the paths: some 12 pins are tied into the scan system. However it is unfair to classify all of these as testing overheads, as 3 major paths of the total 8 are used for functional purposes as well. About half of the scan I/O is attributable to testing alone. One further pin is used to output the global NOP line. However, while there is an increase in pads and bonding from this extra I/O, care has been taken to ensure that the pin count does not increase past 64, the limit for DIL packaging. (The alternative, a pin grid array, is considerably more expensive.)

In the control area, a small amount of extra logic is incurred in adding three extra states to the incompletely coded LID Decoder, to allow LI operations to address the registers of the Output Processor. In each of the ALU Decoders, another three states are added to the incompletely coded set of ALU instructions, to allow the required TEST multiplication and

division instructions. This incurs a small increase in the logic implemented in each.

The modifications to the multiplier/dividers require the addition of one latch, and the alteration of two logic gates from two-input to three-input, in each module.

Taken all together, the overhead for testability is surprisingly low: in terms of transistor count, and given a size estimate of 200,000 transistors for the whole chip, the test overhead is only about 3.5%.[†] This compares well with some of the overhead figures seen in discussions on testability. In particular, some proponents of built in self test claim that area overhead figures of up to 25% are still acceptable. The single most striking increase is in the pin count, and this a matter of design choice: if a reduced pin count was considered important, then all scan path elements could be connected into one path, incurring only three extra pins instead of twelve. (This would, of course, incur a vast increase in the total test time for the chip.)

5.3 Some General Observations.

Perhaps the biggest time and effort saving feature in the design of these tests is the design style used in the modules. Almost all modules are constructed as regular arrays of small, easily testable bit-slice cells, with the notable exceptions being the memory decoders, and they are treated as part of the memory array testing problem rather than as a logic circuit problem. This regularity and the use of small leaf cells allows the test patterns for the modules to be determined simply by inspection, along with the sensitising

[†] Although the chip layout has not been completed, a very conservative first order estimate of the area required for this dedicated test hardware (including all I/O pads, clock and signal routing, clock drivers, signal buffers, scan path cells, extra latches in the multiplier/dividers and extra decoding of the ALU and LID fields), indicates an area overhead of less than 7% of the unaugmented chip area, or less than 6.5% of the total chip area.

paths where required.

Another big bonus is the way the architecture partitions the various modules, requiring simultaneous operations in each with no interaction. This is precisely the condition that allows testing of one partition independently of another, thereby allowing an exponential decrease in the test complexity.

Yet another major factor is the parallel accessibility of the inputs and outputs of the data handling elements: once the Output Processor is verified, all of the data handling elements can be treated as separate partitions, each with independent controllable inputs and observable outputs, and none of them is particularly complex (with the exception of the multiplier/dividers, which are readily altered to regain that simplicity).

The testing of the Control Core is facilitated by the extensive scan paths through it, and it give rise to the observation that, if certain circuits are inter-related, then the more of the inputs or states to those sections that are controllable, the more easily can the tester not only test the whole, but do so using a reduced test set, due to the ability to control the dependence of one section upon another.

5.4 Further Work.

Many of the tests are specified purely in terms of the input patterns that are required to sensitise particular pathways or to create particular states within the partitions under test. In the actual test situation, the output patterns from these tests are required as pre-computed values, against

which to compare the response of the circuit. Thus some work must be done to convert the input pattern specifications into expected responses. In most cases this is trivial, but the ASSAs and multiplier/dividers all perform additions on their test patterns, so the expected response must be calculated.

There is no readily amenable method of calculating the fault cover of the test set available within the design environment at the moment. However, should one become available, it should prove interesting to extract gate level models of the circuit partitions and to check the actual fault cover of the tests specified in Appendix 1.

5.5 Conclusion.

A test procedure has been designed to test the TFB chip. It involves not only the external application of test vectors, but also the inclusion of certain circuit elements to enhance or enable the testing. Both the internal circuitry and the external test procedure are detailed within this work.

In the final analysis, the test length is not inordinately long, and the area and complexity overhead for the test circuit is remarkably low, so it is fair to say that an acceptable compromise has been reached between system functionality, test time and area and complexity overheads.

Appendix A

TFB Test Specification.

This appendix contains the specification of the complete TFB chip test. It is broken up into eight main sections, each dealing with the tests for a particular partition. These sections correspond to the identically titled sections of Chapter 4, in which some explanations of the tests are made. In this appendix, notes and comments will be kept to a minimum. It is assumed that the reader is well versed in the architecture, and to a certain extent the layout of the chip. In general, instructions and pin settings will be detailed as and where necessary, and it may be assumed that if a setting is not specified, then it is either obvious from the context, or it does not matter. The notation used for instruction fields is generally in accordance with that developed in the TFB Working Specification, which will not be produced herewith, as most of the notation used is expanded or explained in the discussions in Chapter 4, if not in this Appendix.

Hexadecimal notation is frequently used herein to represent bit patterns, and is represented by the form $\langle xxxx \rangle$ where the leftmost digit

is the most significant, the rightmost the least. The number of digits will correspond to the number of bits being represented, except where the bit pattern lengths are too large, as in the 64 bit Write Register, or are not multiples of four. In the latter case, the notation will add leading bits to fill the patterns out to multiples of four, and these fill bits will be set to zero. In the former case, enough of a pattern will be given to indicate the required value. For example, a 64 bit alternating one and zero pattern is represented as $\langle AA...A \rangle$. Where binary patterns must be explicitly indicated, notation such as (1011) will be used.

For the sake of brevity, the following abbreviations will be used:

C1L Control Scan Path 1 Lower

C1U Control Scan Path 1 Upper

C2L Control Scan Path 2 Lower

C2U Control Scan Path 2 Upper

DecA Decoder Scan Path A,B Upper

DecB Decoder Scan Path A,B Lower

DecC Decoder Scan Path C,D Upper

DecD Decoder Scan Path C,D Lower

PSR Process Status Register

WR Write Register

IR Instruction Register

BAR n Bad Address Register n

EnBAR n Enable bit for Bad Address Register n

WR-CLR Write Register Clear Control Input bit

WR-SET Write Register Set Control Input bit

WP-IPC Write Pointer Select IPC Control Input bit

WP-JPC Write Pointer Select JPC Control Input bit

WP-BAR0 Write Pointer Select BAR0 Control Input bit

WP-BAR1 Write Pointer Select BAR1 Control Input bit

WP-BAR2 Write Pointer Select BAR2 Control Input bit

WP-BAR3 Write Pointer Select BAR3 Control Input bit

With regards to the scan paths, these are scanned by toggling the appropriate scan clock line. When a scan clock is not specifically required to enable a path in the tests, it is held low. Where possible, scan paths are to be clocked in parallel: this is implicit, and will not generally be pointed out in every instance. As the test is designed to be sequential, control pins are to be held at a specified values until otherwise specified.

A.1 The Control Core.

A.1.1 Start Up Tests.

On start-up, a chip is usually subjected to a number of analogue tests prior to commencement of the digital testing. These have not been considered in this report, although they are of some importance in the production environment. One applicable test is to monitor the supply current as power-up occurs, and in the idle time thereafter: for CMOS, the quiescent current should be very small, and any wild excursions during power-up may indicate shorted lines or stuck-at transistors.

The first digital test applied is part of these start-up tests: on power-up, the chip should have Single Step mode selected, system clocks running, scan clocks held low, Single Step Strobe held low, and Write held low. After the system has settled, the NOP pin should have settled high. If it has not, the rest of the test is not valid, because all the scan paths will continuously be overwritten by clocked latches not disabled.

A.1.2 Scan Path Verification.

This section scans through API test patterns to validate the scan path shift and hold capabilities, lengths and thus the path multiplexing and selection.

Keep Single Step mode selected, Single Step Strobe and Write held low, system clocks running. Scan clocks and selects activated as directed.

Select DecA Scan Path (use Scan A,B pins/Scan Clock 1/Decoders/Upper), scan in repetitions of hexadecimal pattern `< 0FA >` until the first distinct `< 0FA >`

pattern appears at the output.

Repeat for DecC Scan Path (Scan C,D pins/Scan Clock 2/Decoders/Upper).

Repeat this sequence for DecB and DecD Scan Paths, using the controls as above, except with Lower selected at the pins, instead of Upper.

Then holding select at Decoders/Upper, scan ones into both DecA and DecC Scan Paths using both scan clocks in parallel.

Check to see $\langle 0FA \rangle$ pattern appears, until replaced by $\langle FFF \rangle$.

Counting the scan verifies path length.

Repeat this for DecB and DecD Scan Paths (control identical except select Decoders/Lower).

This completes the Decoder Scan Paths test. A similar procedure is used to test the Control Scan Paths.

Select C1U (Control/Upper/Scan Clock 1/Control Scan 1 pins).

Scan in $\langle 0FA \rangle$ pattern repetitions until first distinct $\langle 0FA \rangle$ appears at output.

Repeat for C2U (Control/Upper/Scan Clock 2/Control Scan 2 pins).

Repeat for C1L (Control/Lower/Scan Clock 1/Control Scan 1 pins).

Repeat for C2L (Control/Lower/Scan Clock 2/Control Scan 2 pins).

Select C1U and C2U using both scan clocks in parallel to scan all ones in, except for a zero in the bit position of the C2U pattern that will correspond to WR-SET.

Check to see $\langle 0FA \rangle$ pattern appears, until replaced by $\langle FFF \rangle$.

Counting the scan verifies path length.

Repeat this for C1L and C2L (control identical except select Control/Lower), with different data. Use $\langle 000 \rangle$ for C1L, leaving the column multiplexors connecting each CS column to one IR cell (no sparing). Use all ones for C2L.

This completes the Scan Paths' verification.

A.1.3 Write, Wordline Latches

All controls are as for previous test, all scan clocks held low. The previous test leaves ones everywhere in C2L, C2U and C1U, except in WR-SET. All six Write pointer (WP) selects are therefore enabled, as are the BARs. All six contain the same address (all ones).

Set the Write line for one cycle, then set low. Wait two cycles.

Scan out C2L observing all wordline latches set to zero by Write, except for the four corresponding to the BARs.

Scan out C1U to check that WR was cleared successfully.

A.1.4 Main CS Array Decoding, Wordlines, Write Pointer Select.

Scan in C2U: set WP-IPC, clear all other WPs, IPC = < 00 >, JPC = < 7F >, disable all BARs, set WR-SET, clear WR-CLR. (Merge this scan with last of previous test.)

Write (set Write pin high for one clock cycle).

Scan out C2L, check correct and unique wordline selected by IPC, scan in all ones.

Scan in C2U as above except: set WP-JPC and WR-CLR, clear WP-IPC and WR-SET.

In parallel, scan out C1U to check operation of WR-CLR in Write: this is needed once only.

Write.

Scan out C2L, check correct and unique wordline selected by JPC, scan in all zeros.

Scan in C2U to set WP-IPC and WR-CLR for next Write.

The above sequence of two writes and wordline latch examinations should be carried out for all values of IPC and JPC, and this is best accomplished by incrementing IPC and decrementing JPC between each execution of the sequence, and repeating the procedure until $IPC = \langle 7F \rangle$ and $JPC = \langle 00 \rangle$ have been tested. Note that the scan out of C1U to check correct WR-SET operation is required once only. After the initial sequence, all Writes use WR-CLR, resulting in the array being completely cleared by the end of the test. The alternating one or zero refill of C2L ensures that each wordline is tested for its ability to set and clear the wordline latches. This procedure tests all of the main array pointer decoding and wordlines.

A.1.5 BAR Wordlines and Addressing

This section of test first verifies the BAR wordlines, next the matching to IPC and JPC and the consequent main array pointer override, then the rest of the bitwise matching test patterns are applied to all BARs in parallel, followed by a test set of non-matching patterns.

Scan in C2U: set WP-BAR0, WP-IPC, WP-JPC, WR-CLR; $IPC = \langle 70 \rangle$, $JPC = \langle 0F \rangle$, all BARs set to $\langle 00 \rangle$; clear all EnBARs. (Merge with previous test last scan.)

Write.

Scan out C2L: refill with ones; check unique and correct BAR0 wordline, JPC override, IPC override.

Repeat, substituting WP-BAR1 for WP-BAR0, then again for WP-BAR2 and WP-BAR3.

This completes testing of BAR wordline drive capability.

Scan in C2U: set WP-IPC, EnBAR0, WR-CLR; BAR0=IPC=< 00 >, JPC=< 7F >, all other BARs set to < 7F >; clear all other EnBARs.

Write.

Scan out C2L: refill with ones; check correct BAR0 wordline to flag match, IPC override.

Scan in C2U: set WP-JPC, EnBAR0, WR-CLR; BAR0=JPC=< 00 >, IPC=< 7F >, all other BARs set to < 7F >; clear all other EnBARs.

Write.

Scan out C2L: refill with ones; check correct BAR0 wordline to flag match, JPC override.

Repeat, substituting EnBAR1 for EnBAR0 and BAR1 for BAR0, then again for BAR2 and BAR3.

This completes the individual BAR match-override test. Match and mismatch can now be checked with all BARs in parallel.

Scan in C2U: set WP-IPC, all EnBARs, WR-CLR; all BARs=IPC=< 7F >, JPC=< 00 >.

Write.

Scan out C2L: refill with zeros; check matches, IPC override.

Repeat, substituting IPC for JPC and vice versa.

Repeat both above, substituting < 2A > for < 7F > in pointers, then again substituting < 55 > for < 2A >.

This completes the test for correct bitwise matching. Now test for mismatches.

Scan in C2U: set WP-IPC, all EnBARs, WR-CLR; all BARs=< 00 >, IPC=< 7F >, JPC=< 2A >.

Write.

Scan out C2L: refill with zeros; check NO matches, NO IPC override.

Repeat, substituting IPC for JPC and vice versa.

Repeat both above, substituting < 7F > for < 00 > in the BARs, and substituting < 00 > for < 7F > in JPC or IPC (whichever is selected).

Repeat the sequence of four tests above 14 more times, each time substituting a pair of hex addresses from the following set for (< 00 >, < 7F >).

| Hexadecimal Number Pairs | | | | | | | | | | | | | |
|--------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 40 | 20 | 10 | 08 | 04 | 02 | 01 | 3F | 5F | 6F | 7F | 7B | 7D | 7E |
| 00 | 00 | 00 | 00 | 00 | 00 | 00 | 7F |

This set of mismatch tests completes the BARs' testing.

A.1.6 CS Column Faults, IPC Incrementer.

This section aims to test for column SA faults, and the associated paths for SA and API faults. Merged with these are tests of the column multiplexor chain, IPC incrementer tests, and incomplete tests of the conditional evaluation circuitry. The previous test sequences have left the CS array filled with zeros.

Scan in C2U: IPC=< 7F >, set WP-IPC. (Initialisation.)

Single Step Strobe: observe NOP pin; should go low for one cycle.

Scan C2U: out; IR all zero, IPC=< 00 >: in; WP-IPC and WR-SET set,
IPC=< 00 >.

Scan C1U: in; fill PSR with zeros.

Write. (Write WR to memory using IPC value.)

Single Step Strobe.(Read word from CS using IPC, increment IPC.) NOP set.

Scan C2U: out; IR all ones, IPC=< 01 >: in; WP-IPC and WR-SET set,
IPC=< 7E >.

Scan C1U: in; fill WR with < AA...A >, fill PSR with zeros, except for a single
one in the position matching the WAIT bit for the instruction in IR.

Write.

Single Step Strobe. NOP low for 1 cycle.

Scan C2U: out; IR=< AA...A >, IPC=< 7F >: in; WP-IPC and WR-SET set,
IPC=< 7E >.

Scan C1U: in; fill PSR with zeros, except for a single one in the position
matching the WAIT bit for the instruction expected in IR.

Scan C1L: in; all ones (swap multiplexors to other position).

Single Step Strobe. NOP low for 1 cycle.

Scan C2U: out; IR=< 55...5 >: in; WP-IPC, IPC=< 03 >. (This address should
still contain all zeros.)

Single Step Strobe. NOP low for 1 cycle.

Scan C2U: out; IR=< 00..01 >, IPC=< 04 >: in; WP-IPC and WR-SET set,
IPC=< 07 >.

Write.

Single Step Strobe. NOP low for 1 cycle.

Scan C2U: out; IR=< FF...F >, IPC=< 08 >: in; WP-IPC set, IPC=< 01 >.

Scan C1U: in; WR=< 55...5 >, fill PSR with ones, except for a single zero in the

position matching the WAIT bit for the instruction in IR.

Write.

Single Step Strobe. NOP remains high.

Scan C2U: out; IR=< AA..AB >, IPC=< 02 >: in; IPC=< 01 >.

Scan C1L: in; all zeros (swap multiplexors back to original position).

Scan C1U: in; fill PSR with ones, except for a single zero in the position matching the WAIT bit for the instruction in IR.

Single Step Strobe. NOP remains high.

Scan C2U: out; IR=< 55...5 >: in; IPC=< 0F >.

Single Step Strobe.

Scan C2U: out; IPC=< 10 >: in; IPC=< 1F >.

Single Step Strobe.

Scan C2U: out; IPC=< 20 >: in; IPC=< 3F >.

Single Step Strobe.

Scan C2U: out; IPC=< 40 >.

The IPC incrementer has been verified. Column faults have been tested. Column multiplexors verified. Some conditional control verified.

A.1.7 CS Cell Tests.

Using a subset of the MARCH test, the cells are tested for SA and API faults involving nearest column neighbours. The array is initialised to zeros, by resetting to zero any rows written with non-zero values in the last section: all others were reset previously and should not have changed. First a zero to one transition is marched up the array, setting the complete array

to one, then a one to zero transition is marched up. Note that physically BAR0 is at the bottom edge of the array, with BAR3 adjacent to Row0 of the main array, while Row127 is at the very top edge of the array. Between the named rows, the addressing proceeds in a linear manner. Also, the first and last rows of the array (in this case, BAR0 and Row127) have reduced tests, because they each have only one neighbour capable of being affected by a fault.

Scan C2U: in; WP-IPC and WR-CLR set, IPC=< 7E >. Write.

Single Step Strobe.

Single Step Strobe. (Cycles IPC to desired value faster than scan.)

Write. (IPC=< 00 >, Control inputs as before.)

Single Step Strobe. (Cycles IPC to < 01 > faster than scan.)

Write.

Single Step Strobe.

Single Step Strobe. (Cycles IPC to < 07 > faster than scan.)

Write. (IPC=< 07 >, Control inputs as before.)

This completes the re-initialisation.

Scan C2U: in; BAR0=IPC=< 00 >, set EnBAR0.

Single Step Strobe.

Scan C2U: out; IR=< 00...0 >: in; set WR-SET, WP-BAR0, EnBAR0,

IPC=BAR0=< 00 >

Write. (Write ones to BAR0)

Single Step Strobe.

The preceding instructions are the reduced test for the bottom row.

Scan C2U: out; IR=< FF...F >: in; BAR1=IPC=< 01 >, set EnBAR1.

Single Step Strobe.

Scan C2U: out; IR=< 00...0 >: in; set WR-SET, WP-BAR1, EnBAR1,

IPC=BAR1=< 01 >

Write. (Write ones to BAR1)

Single Step Strobe.

Scan C2U: out; IR=< 11...1 >: in; set WP-BAR0, EnBAR0,

IPC=BAR0=< 00 >

Single Step Strobe.

The last seven instructions form the typical row test set. By replacing BAR n with BAR $n+1$ in successive executions of this test set, all the BARs may be tested, and then by replacing the BARs with row addresses (simply clear all EnBAR bits in Control Input scan) and incrementing the row addresses between each iteration, starting from Row0 (with BAR3 enabled as a column neighbour) and progressing through to Row127, the main array may be tested. The complete sequence (excluding the re-initialisation instructions) may then be repeated, exactly as written above except that every WR-SET is replaced with a WR-CLR. The end status is IPC=< 00 > and the whole array is written with zeros. There is now enough information available to allow the CS to be reconfigured. This is achieved simply by

scanning into C1L an appropriate bit pattern, and scanning into C2U the appropriate BAR values and EnBAR bits: note that this latter operation must be incorporated into any further C2U scans to maintain the correct values for sparing.

A.1.8 Remainder of Control Core.

Still to be verified are parts of the conditional control circuitry, the LID Decoder, the CM register load and hold capabilities, the Loopcounters' load, hold decrement and flag capabilities, and the actual execution of jumps. Again, some of these tests are merged together into single instruction sequences.

Scan C2U: in; set WP-IPC, IPC=JPC=< 00 >.

Scan C1U: in; WR=(LI/Conditional Control CC=(000)/LID=JPC/LI data=< 2A >), PSR all zeros.

Write.

Single Step Strobe. NOP low.

Scan C1U: out; LID Dec. Outputs all zero.

Scan C2U: out; JPC=IPC=< 2A >.

Repeat sequence above exactly as written except as Op, not LI.

Repeat sequence above exactly as written except with CC=(111).

Repeat sequence above exactly as written except with LI data=< 55 >.

This completes test of LID=JPC; execution test later to test timing.

Scan C1U: in; WR=(LI/CC=(001)/LID=Loopcounter0/LI data=< FF >), PSR all ones except for a zero matching WAIT bit.

Write.

Single Step Strobe. NOP high.

Scan C1U: out; LID Dec. Outputs all zero.

Repeat sequence exactly as above except: PSR all zero except for single one matching WAIT bit, LI Data=< 00 >.

Repeat sequence exactly as original except Op, not LI, and LI Data=< FF >.

This concludes test of CC=(001) state, and of LID=Loopcounter0.

Repeat sequence of four tests, substituting CC=(110), and Loopcounter1.

Repeat sequence of four tests, substituting CC=(100), and Loopcounter2.

Repeat sequence of four tests, substituting CC=(011), and Loopcounter3.

This concludes tests of the three conditional control states, and of LID=Loopcounter n . Next test the LID=Accumulator latch lines.

Scan C1U: in; WR=(LI/CC=(000)/LID=LWRACCA). (LI low byte AccA)

Write.

Single Step Strobe. NOP low.

Scan C1U: out; LID Dec. Outputs all zero except for LWRACCA.

Repeat sequence as above except CC=(111),PSR all zero.

Repeat both test sequences for LID=HWRACCA (LI high byte AccA).

Repeat both test sequences for LID=BWRACCA (LI both bytes AccA).

Repeat all six test sequences for each of Accumulators B,C and D.

Repeat the two original sequences for LID=BALAC (LI both bytes, all accumulators).

Scan C1U: in; WR=(Op/CC=(000)/LID=BALAC).

Write.

Single Step Strobe. NOP low.

Scan C1U: out; LID Dec. Outputs all zero under Op.

This completes Accumulator Latch lines LID testing.

Scan C1U: in; WR=(LI/CC=(000)/LID=IO Control).

Write.

Single Step Strobe.

Scan C1U: out; LID Dec. Outputs all zero except for IO Control.

Repeat above test, changing only LI to Op.

Repeat above test, substituting CC=(111) and PSR all zero.

This completes tests of LID=IO Control.

Repeat above three tests, substituting LID=OutReg1.

This completes tests of LID=OutReg0.

Repeat above three tests, substituting LID=OutReg0.

This completes tests of LID=OutReg1.

Repeat above three tests, substituting LID=CM, LI data = all zeros except for first arithmetic flag. In CC=(111) case, put PSR = all ones except first arithmetic flag.

This concludes examination of LID Dec. Outputs: have commenced examination of operation of the TEST circuitry, and of CM loadability, with 1st CM flag demonstrated unique.

Scan C1U: in; WR=(LI/CC=(111)/LID=CM/LI Data=CM for 2nd arithmetic flag). PSR all zero except 1st arithmetic flag.

Write.

Single Step Strobe. NOP low. (TEST verified for 1st flag, 2nd flag mask loaded).

Scan C1U: in; WR=(LI/CC=(111)/LID=CM/LI Data=CM for 3rd arithmetic flag). PSR all ones except 2nd arithmetic flag.

Write.

Single Step Strobe. NOP high.

Scan C1U: in; WR=(LI/CC=(111)/LID=CM/LI Data=CM for 3rd arithmetic flag). PSR all zero except 2nd arithmetic flag.

Write.

Single Step Strobe. NOP low. (TEST verified for 2nd flag, 3rd flag mask loaded).

Repeat tests in the form of the last two above, incrementing along CM till 16th flag tested. The success is used to load the next flag mask CM = all ones.

Scan C1U: in; WR=(LI/CC=(111)/LID=CM/LI Data= < 0...0 > (no flags)).

PSR=< A50F >.

Write.

Single Step Strobe. NOP low. (TEST verified for multiple flag, empty flag mask loaded).

Write. (Same WR, but CM already loaded with all zeros.)

Single Step Strobe. NOP high.

This concludes the verification of the TEST state, and of the CM and the conditional evaluation circuitry. It remains to verify that the JUMP instruction (LI JPC) works correctly as far as timing is concerned: it has already been logically checked in Single Step Execution. Also the load, decrement and flag capabilities of the Loopcounters still require verifica-

tion, and this may only be done by inference, so an executing program is the quickest way to test them. First consider the JUMP test. Load a short program, set IPC to the start address, and switch the mode pin to Program Execute. The NOP pin is used to flag internal operations, and its periodicity allows the program action to be deduced. The CM is already loaded with all zeros from the previous test, allowing TEST to flag with NOP high.

Scan C1U: in; WR=(LI/UNCON/LI JPC/LI Data= < 00 >)

Scan C2U: in; set WP-IPC, IPC=< 01 >.

Write.

Scan C1U: in; WR=(Op/UNCON/NULL)

Scan C2U: in; set WP-IPC, IPC=< 02 >.

Write.

Scan C1U: in; WR=(LI/TEST/LI JPC/LI Data= < 00 >) PSR=(00...0)

Scan C2U: in; set WP-IPC, IPC=< 00 >, JPC=< 2A >.

Write.

Having verified the jumps, it is possible to use them in a program that tests the Loopcounters' decrement capability by simply looping until the Loopcounters reach zero: this operation exhaustively tests the Loopcounter decrement, and a successful result implies a correct load. It is quickest and easiest to load one simple program and then customise it to suit the

successive counters under test.

Scan C1U: in; WR=(LI/UNCON/LI CM/LI Data= CM for Loopcounter0)

Scan C2U: in; set WP-IPC, IPC=< 01 >.

Write.

Scan C1U: in; WR=(LI/TEST/LI JPC/LI Data=< 02 >)

Scan C2U: in; set WP-IPC, IPC=< 02 >.

Write.

Scan C1U: in; WR=(LI/UNCON/LI JPC/LI Data= < 02 >)

Scan C2U: in; set WP-IPC, IPC=< 03 >. Write.

Scan C1U: in; WR=(LI/UNCON/LI Loopcounter0/LI Data=< FF >)

Scan C2U: in; set WP-IPC, IPC=< 00 >.

Write.

After running the program once, stop it by selecting Single Step mode, scan in a new start value for the Loopcounter by re-entering the instruction in address < 00 > with new LI Data: < AA > for first substitution, < 55 > for second. Having run this program three times for the same counter, change the CM in the instruction in address < 01 > to select first Loopcounter1, then 2 and 3, repeating the three different program runs for each. This completes the Loopcounters test, and with it the Control Core test.

A.1.9 Test Times.

| Part | Elements Tested | Σ_{cycles} |
|---------------------------------------|----------------------------------|-------------------|
| 1a | Decoder Scan Paths | 200 |
| 1b | Control Scan Paths | 722 |
| 2 | Write latching | 136 |
| 3 | Main Array Wordlines | 48384 |
| 4 | BAR function | 14742 |
| 5 | Column faults, IPC increment | 1363 |
| 6 | CS Cell Test | 54122 |
| 7a | Control Core — Single Step | 8064 |
| 7b | Control Core — Program Execution | 4081 |
| Control Core Test: Total Clock Cycles | | 131814 |

With the chip running at design clock speed (8 MHz) the Control Core test will take about 16.5 milliseconds to carry out.

A.2 The ALU and Data Memory Decoders.

The combinational decoders are tested exhaustively over the range of operational microcode input combinations, and pseudo-exhaustively to show that the LI instructions disable the decoder outputs. The tests are all applied by scanning an instruction into the WR, Writing, then Single Stepping once to clock the instruction through to the combinational decoder output latches, and finally scanning out. The scan out of DecA and DecC may be done in parallel, as may the DecB and DecD. The scan in of the instruction is done by scanning C1U and C2U in parallel. In each case the test sequence is of the following form:

Scan C1U: in; WR=(Instruction specified in following subsections.)

Scan C2U: in; set WP-IPC, IPC=< dont care >.

Write.

Single Step Strobe.

Scan DecA and DecC in parallel.

Scan DecB and DecD in parallel.

Each test requires 64 cycles scan in, 3 Write cycles, 3 cycles for the Single Step, and two 48 cycle scans out, resulting in a count of 118 cycles per test.

A.2.1 Operation States Decoding.

To verify the decoding of the instruction microcode into control lines under an Operation instruction, the decoders are exhaustively tested by repeating the test sequence above for the test instruction template below, with the values of $(A_4, A_3, A_2, A_1, A_0)$ cycled through all 32 permutations.

Op UNCON $\underbrace{(A_4, A_3, A_2, A_1, A_0) \times 4}_{\text{ALU Fields}}$ $\underbrace{(A_3, A_2, A_1, A_0) \times 4}_{\text{DM Fields}}$ $\underbrace{(LID = BALAC)}_{\text{IO Fields}}$ $\underbrace{(BREAK) \times 10}_{\text{Switch Fields}}$

A.2.2 LI Disabling.

To ensure that the decoder outputs are not activated by LI Data words in LI instructions, the test sequence is repeated, using the instruction template below, with the LI Data and spare bits such that together they comprise the same bit patterns as four identical ALU Operation fields filled with one of the set of eight test instructions. The ALU instructions are chosen so

that exercising the complete set in Operations has the effect of setting all the decoder outputs at least once. The point of the test is to ensure that the LI instruction overrides the operational decoding.

LI UNCON $(\underbrace{D_{15}, D_{14}, \dots, D_0}_{\text{LI Data}}, \underbrace{S_3, \dots, S_0}_{\text{Spares}})$ $(\underbrace{\text{XXXX}}_{\text{DM Fields}}) \times 4$ $(\underbrace{\text{NULL}}_{\text{LID}})$ $(\underbrace{\text{BREAK}}_{\text{Switch Fields}}) \times 10$

with $(\underbrace{D_{15}, D_{14}, \dots, D_0}_{\text{LI Data}}, \underbrace{S_3, \dots, S_0}_{\text{Spares}})$ set equal to $(\underbrace{A_4, A_3, A_2, A_1, A_0}_{\text{ALU Fields}}) \times 4$,

and the $(A_4, A_3, A_2, A_1, A_0)$ being chosen from the following list of instructions.

TEST-MVDDGO multiplier/divider lines, accumulator lines

ADDAX multiplier/divider lines, adder lines

LOADX multiplier/divider load line

BWRACC accumulator lines

SHRACC adder shift line

SHLACC adder shift line

NEGACC adder lines

A.2.3 LI Selection of Accumulators.

The LI instruction may select the Accumulator latch lines, and thus they are tested for correct operation, with a single test, of the same form as those in immediately preceding test section, but with the LI Data such that the microcode for the NULL Operation is applied to the decoder inputs, and the LID set to BALAC, which selects all accumulator latch lines simultaneously.

A.2.4 Test Length.

As noted earlier, each test of this section takes 118 cycles to load, execute and read out, and there are forty such tests required, giving rise to a total test length of 4720 cycles, which at design clock speed takes 0.59 milliseconds.

A.3 The Output Processor.

This test requires a certain degree of synchronism to be maintained between the exterior tester and the internal program in order that the OREQ pin may be toggled at the appropriate times, and the NOP pin may be interpreted correctly. Thus the test is specified in terms of the program and the concurrent action at the pins, both inputs required and output expected. Each ruled line represents the start of one clock cycle.

The following abbreviations are used to allow compact notation:

phiN System Clock phase *N*

DS "Data Sent" flag

SO Strobe Out Pulse(s)

NO SO No Strobe Out Pulse

WDS WAIT "Data Sent" Condition Control

LI Load Immediate Instruction

LI OR0 LI Output Register 0

LI OR1 LI Output Register 1

LI IO M_n LI I/O mode *n*

The table which follows details the relationship between the internal program instructions, their execution status, and the activity at the external pins, both input and output.

| Program | Pins |
|---------------------|--|
| LI IO M6 | |
| WDS LI OR0 < 00FF > | |
| WDS LI OR1 < FF00 > | |
| WDS NULL | NOP HI |
| <i>program idle</i> | OREQ HI phi2, phi3 NOP HI |
| <i>program idle</i> | OREQ HI phi1, LO phi2 (<i>abort</i>) NOP HI, SO phi2, phi3 (OR0,OR1)=< 00FF >, then < FF00 > |
| <i>program idle</i> | NOP HI |
| <i>program idle</i> | OREQ HI phi1, phi2 NOP HI, SO phi2, phi3 (OR0,OR1)=< 00FF >, then < FF00 > |

| | |
|---------------------|---|
| <i>execute</i> | NOP LO |
| NULL | OREQ HI phi1, phi2 NOP LO, NO SO |
| <hr/> | |
| WDS LI OR0 < FF00 > | OREQ HI phi1, phi2 NOP LO, NO SO |
| <hr/> | |
| WDS LI OR1 < 00FF > | OREQ HI phi1, phi2 NOP LO, SO phi2, phi3 (OR0,OR1)=< FF00 >, then < 00FF > |
| <hr/> | |
| WDS LI OR1 < 55AA > | OREQ HI phi1, phi2 NOP LO, NO SO |
| <hr/> | |
| WDS LI OR0 < AA55 > | OREQ HI phi1, phi2 NOP LO, SO phi2, phi3 (OR0,OR1)=< AA55 >, then < 55AA > |
| <hr/> | |
| LI OR0 < 55AA > | OREQ tied HI NO SO |
| <hr/> | |
| LI OR1 < AA55 > | OREQ tied HI (<i>ends API test</i>) SO phi2, phi3 (OR0,OR1)=< 55AA >, then < AA55 > |
| <hr/> | |
| WDS LI OR0 < 0000 > | |
| <hr/> | |
| WDS LI OR1 < FFFF > | (<i>clears DS flag</i>) |
| <hr/> | |
| LI IO M0, EnL LO | OREQ tied HI |

SO phi2, phi3 (*DS not set*)
(OR0,OR1)=< 00FF >, then < 00FF >

WDS LI OR0 < 0000 >

WDS LI OR1 < FFFF > (*clears DS flag*)

LI IO M0, EnL HI OREQ tied HI
 NO SO (*DS set by LI IO*)

WDS LI OR0 < FF00 > OREQ HI
 NO SO

WDS LI OR1 < AA55 > OREQ HI
 SO phi 2
 (OR0,OR1)=< FF55 >

WDS LI OR1 < FA05 > OREQ HI
 NOP LO, NO SO

WDS LI OR0 < 50AF > OREQ LO
 NOP LO, NO SO

WDS LI IO M2 OREQ HI phi1, phi2
idling NOP HI, SO phi 2
 (OR0,OR1)=< FFAA >

execute NOP LO

WDS LI OR1 < 55AA > OREQ HI
NOP LO, NO SO

WDS LI OR0 < FF00 > OREQ HI
SO phi 2, phi3
(OR0)=< FF > then < 00 >

LI IO M4 NOP LO

WDS LI OR0 < 00FF > OREQ HI
NOP LO, NO SO

WDS LI OR1 < AA55 > OREQ HI
SO phi 2, phi3
(OR1)=< AA > then < 55 >

LI IO M1 NOP LO

WDS LI OR1 < 55AA > OREQ HI
NOP LO, NO SO

WDS LI OR0 < FF00 > OREQ HI
SO phi 2
(OR0)=< FF >

LI IO M3 NOP LO

WDS LI OR0 < 00FF > OREQ HI
NOP LO, NO SO

| | |
|---------------------|--------------|
| WDS LI OR1 < AA55 > | OREQ HI |
| | SO phi 2 |
| | (OR1)=< AA > |

| | |
|---------|--------|
| NULL Op | NOP LO |
|---------|--------|

| | |
|------------------|-------------------|
| LI JPC (to NULL) | <i>(end loop)</i> |
|------------------|-------------------|

The test requires 35 instructions to be loaded, each requiring 67 cycles to scan in and Write, plus about 55 cycles of execution time, resulting in a test length of around 2400 cycles, or at 8MHz, 0.30 milliseconds.

A.4 The Ring Bus.

This section of the test is conducted purely as an internally controlled program execution. The Output Processor is used to pass results off-chip for external analysis. This is arranged by loading the Output Processor control to select I/O mode 5, a synchronous 16 bit 1 word transfer, and tying the OREQ pin high, thus allowing the Processor to send data as it is received from the internal modules. Unless otherwise noted, all instruction fields not specified are assumed to be NULL (resulting in no action). All switch settings are assumed to be at BREAK unless specifically noted otherwise in the instruction. The following table contains abbreviations used to allow compact notation.

RP n Im x Read DM x, pointer $n = 0,1$, increment by $m = 0,1$ or
P=preset.

WP n Im x Write DM x, pointer $n = 0,1$, increment by $m = 0,1$ or
P=preset.

Px Pass, 16 bit Pass/Break switch on quadrant x.

Bx Break, 16 bit Pass/Break switch on quadrant x.

Pxy Pass, 32 bit Pass/Break switch between quadrants x and y.

Pall Pass, all Pass/Break switches.

Bxy Break, 32 bit Pass/Break switch between quadrants x and y.

PLx Pass Lower Byte, T-switch on quadrant x.

PLall Pass Lower Byte, all T-switches.

PUx Pass Upper Byte, T-switch on quadrant x.

PUall Pass Upper Byte, all T-switches.

BLx Break Both Bytes, T-switch on quadrant x.

HRDACCx High byte, ReaD ACCumulator x.

HWRACCx High byte, WRite ACCumulator x.

LRDACCx Low byte, ReaD ACCumulator x.

LWRACCx Low byte, WRite ACCumulator x.

BRDACCx Both bytes, ReaD ACCumulator x.

BWRACCx Both bytes, WRite ACCumulator x.

OP Operation instruction.

LI LI instruction.

LI OR0 LI Output Register 0.

LI OR1 LI Output Register 1.

LI BOR LI both Output Registers.

WOR0 Write Output Register 0.

WOR1 Write Output Register 1.

WBOR Write both Output Registers.

LI JPC Jump Instruction.

LI IO Mn LI I/O Mode *n*.

This first section of instruction coding is designed to test one of the bus segments immediately adjacent to the Output Processor and LI bus driver segment. This segment connects to the DM data buffer registers, one of which is used in the test. The final verification of the Output Processor under Operation instructions is also carried out.

LI IO M5

LI WP0I0 A < 0000 > PA (*load closest DM*)

LI BOR < FFFF > PA (*reverse charge bus*)

OP RP0I0 A WBOR PA (*read buffer back*)

LI WP0I0 A < FFFF > PA

LI BOR < 0000 > PA (*reverse charge bus*)
OP RP0I0 A WBOR PA (*read buffer back*)
LI WP0I0 A < AAAA > PA (*load closest DM*)
LI BOR < 5555 > PA (*reverse charge bus*)
OP RP0I0 A WBOR PA (*read buffer back*)
LI WP0I0 A < 5555 > PA
LI BOR < AAAA > PA (*reverse charge bus*)
OP RP0I0 A WBOR PA (*read buffer back*)

This completes the verification of the quadrant A bus segment, the data buffer 0 of DM A, and the Pass state of quadrant A Pass/Break switch. It also completes the final (Operation) test of the Output Processor. The data buffer is now used to test the switch for isolation.

LI WP0I0 A < 0000 > PA (*load closest DM*)
LI BOR < FFFF > PA (*reverse charge bus*)
OP RP0I0 A WBOR BA (*try to read buffer back*)
LI WP0I0 A < FFFF > PA
LI BOR < 0000 > PA (*reverse charge bus*)
OP RP0I0 A WBOR BA (*try to read buffer back*)

This completes isolation testing of switch A. Now to test continuity around the Ring.

LI WP0I0 < 0000 > PA
LI BOR Pall PUall < FFFF >
OP HWRACC all WBOR (*output test result*)
LI BOR Pall PUall < FFFF >
OP HRDACC Pall PUall A WBOR (*output AccA high byte*)
LI BOR Pall PUall < FFFF >

OP HRDACC Pall PUall B WBOR (*output AccB high byte*)
 LI BOR Pall PUall < **FFFF** >
 OP HRDACC Pall PUall C WBOR (*output AccC high byte*)
 LI BOR Pall PUall < **FFFF** >
 OP HRDACC Pall PUall D WBOR (*output AccD high byte*)
 LI WP0I0 < 0000 > PA
 LI BOR Pall PLall < **FFFF** >
 OP LWRACC all WBOR (*output test result*)
 LI BOR Pall PLall < **FFFF** >
 OP LRDACC Pall PLall A WBOR (*output AccA low byte*)
 LI BOR Pall PLall < **FFFF** >
 OP LRDACC Pall PLall B WBOR (*output AccB low byte*)
 LI BOR Pall PLall < **FFFF** >
 OP LRDACC Pall PLall C WBOR (*output AccC low byte*)
 LI BOR Pall PLall < **FFFF** >
 OP LRDACC Pall PLall D WBOR (*output AccD low byte*)

This verifies all accumulator bytes not SA1. Repeat test format to test for SA0 by swapping < 0000 > for < **FFFF** > and vice versa. Repeat twice more for API faults, substituting first < 5555 > and then < **AAAA** > for the accumulator value, using the complement to reverse-charge the bus. The reverse charge on the bus serves to provide a worst case load for the bus drivers. The next test section checks the sign extend/zero fill capabilities of the T-switches. The tests are intermeshed so that the output of one test forms the pre-charge or input for another module's test. In this section all of the 16 bit Pass/Break switches are set to Pass, and only the 32 bit Pass/Break and T-switches are manipulated.

LI WP0I0 A < **FFFF** > PA
 LI BOR < 0000 > PUall Pall except BAB

OP RP0I0 BWRACCA,B PLall Pall except BAB
 OP HRDACCA PUA PLC PLD PCD LWRACCs C,D WBOR
 OP HRDACCB PUall PAB PCD BWRACCs C,D WBOR
 OP LRDACCC PCD PLall BAB HWRACCs A,B WBOR
 OP LRDACCD PCD PLC WBOR
 OP HRDACCA PUA WBOR
 OP HRDACCB PUall PAB BCD HWRACCs C,D WBOR
 OP LRDACCA PLall PAB BCD BWRACCs C,D WBOR
 OP HRDACCC PUC PLA PAB WBOR
 OP HRDACCD BAB PCD PUall LWRACCs A,B WBOR
 OP LRDACCA PLall PAB BCD HWRACCs C,D WBOR
 OP LRDACCB PLA PAB WBOR
 OP LRDACCC PLA PLC PAB LWRACCs A,B WBOR
 OP HRDACCC PUall BAB PCD LWRACCs A,B WBOR
 OP HRDACCD PCD PUC WBOR
 OP LRDACCA PLA,B PAB PUC,D BCD LWRACCs C,D WBOR
 OP LRDACCB PAB PLA WBOR
 OP LRDACCC PLC WBOR
 OP LRDACCD PLC PCD WBOR

This concludes the test of the T-switches. The remaining section tests all switches (except Pass/Break switch A) for isolation.

LI WP0I0 A < FFFF >
 OP RP0I0 A PLA BWRACCA BAB
 LI BOR < 0000 > BTA PA BC
 OP BRDACCA BAB BTA WBOR
 LI BOR PUall Pall
 OP BRDACCA BA BAB LWRACCB PUB Pall others WBOR
 OP LRDACCB Pall PLall

LI BOR < 0000 > Pall PLall
 OP BRDACCA BA BTA PAB BTB Pall others WBOR
 LI BOR < 0000 > Pall PLall
 OP BRDACCA BA BTA PAB PLB BB Pall others WBOR
 LI BOR < 0000 > Pall PLall
 OP BRDACCA BA BTA BD PLall Pall others WBOR
 LI BOR < 0000 > Pall PLall
 OP BRDACCA BA BTA BTC PLD BWRACCC Pall others WBOR
 LI BOR < 0000 > Pall PLall
 OP BRDACCA BA BTA BC PLD BWRACCC Pall others WBOR
 LI BOR < 0000 > Pall PLall
 OP BRDACCC BC BTC BCD PUD LWRACCD Pall others WBOR
 OP LRDACCD PC PLC PCD Ball others WBOR
 LI BOR < 0000 > Pall PLall
 OP BRDACCC BC BTC BTD Pall others WBOR

This concludes the Ringbus test. The number of instructions required is 135, plus two LI JPC instructions to form end instructions: two separate program load and execute sequences must be used as the CS can only hold 128 words. The consequent test length is

$$(137 \times (67 \text{ load} + 1 \text{ execute})) = 9316 \text{ cycles,}$$

or at the design clock speed, about 1.17 milliseconds

A.5 The Input Processor.

As for the Output Processor test, this test requires synchronism to be maintained between the internal and external processes to allow correct applica-

tion of test vectors and correct interpretation of results. The test consists of applying data in response to (or contrary to) internally originated requests, and then transferring the contents of the internal bus registers of the Input Processor to the Output Processor, which simply dumps whatever values are written to it to the pins, as its OREQ pin is tied high.

The test is applied as two distinct sections. The first program segment is the synchronous mode test, in which the external tester has no part in the timing of the data transfer, but must simply provide data at the requisite times. The second part requires the external tester to respond to IREQ commands with Strobe pulses and data at the pins at appropriate times, thereby controlling the execution of the internal program indirectly. As this test is concerned more with the correct logical implementation, rather than the analogue performance of the Processor, the Strobe pulses required in the second part are specified to be of the same period as the system clock pulses, and they are required to be applied in phase with the system clocks to allow the internal clock-driven latching to be tested.

The notation used is similar to that used previously for instruction sequences. Each ruled line indicates the beginning of a new clock cycle. The values in the register columns indicate the register contents at the beginning of the particular clock cycle. The values in the Port columns should be applied to the pins for the complete clock cycle in which they are specified. Each value in the Strobe In column indicates a separate pulse, to be applied in synchronism with the system clock phase named. Some new abbreviations and mnemonics are listed below.

IN.INIT Initiate input from the Op microcode.

RIN0 Read input register 0 onto the Ringbus.

RIN1 Read input register 1 onto the Ringbus.

RIN01 Read input half-port registers 0H and 1L onto the Ringbus.

DV "DATA VALID" flag.

WDV WAIT for "DATA VALID" flag.

ϕ_n Clock phase n .

The synchronous test section again uses the double port double byte transfer mode to verify all the data paths in one test sequence and to check the effect of the global NOP on the IREQ line, after which the other two synchronous modes are checked for correct connectivity. As the flag testing is minimal, the program first tests the data paths for connectivity and API and SA faults, using a loop to recursively initiate data transfers until the fault detection set has been processed. On completion of this loop, a test is done of the "abort" procedure to check the effects on IREQ and the data latching. After this the other two modes are checked for correct connectivity by reading data in.

| Strobe | Input Data | | Half-Port Registers | | | | Program Instructions |
|--------|------------|--------|---------------------|--------|--------|--------|----------------------|
| In | Port 1 | Port 0 | Reg 0H | Reg 0L | Reg 0H | Reg 0L | |

LI OPP IO M7

LI IPP IO M6

| | | | | | |
|----------|----------|----------|----------|----------|----------------------|
| | | | | | LI CM=(LOOP0≠0) |
| | | | | | LI LOOP0=3 |
| | | | | | LI WP0I0 < 5500 > |
| $\phi 2$ | < AA > | < 55 > | | | OP IN.INIT RP0I0 |
| $\phi 3$ | < FF > | < 00 > | | | |
| | | | | | @1 OP WDV RIN1 WOR0 |
| $\phi 2$ | α | β | | | OP WDS RIN0 WOR0 |
| $\phi 3$ | γ | δ | | | IN.INIT |
| | χ_1 | χ_2 | χ_3 | χ_4 | LI TEST LOOP0 JPC @1 |

1. $\alpha = \langle FF \rangle$, $\beta = \langle 00 \rangle$, $\gamma = \langle AA \rangle$, $\delta = \langle 55 \rangle$,
 $\chi_1 = \langle FF \rangle$, $\chi_2 = \langle AA \rangle$, $\chi_3 = \langle 00 \rangle$, $\chi_4 = \langle 55 \rangle$.
2. $\alpha = \langle 55 \rangle$, $\beta = \langle AA \rangle$, $\gamma = \langle 00 \rangle$, $\delta = \langle FF \rangle$,
 $\chi_1 = \langle 55 \rangle$, $\chi_2 = \langle 00 \rangle$, $\chi_3 = \langle AA \rangle$, $\chi_4 = \langle FF \rangle$.
3. $\alpha = \langle 00 \rangle$, $\beta = \langle FF \rangle$, $\gamma = \langle 55 \rangle$, $\delta = \langle AA \rangle$,
 $\chi_1 = \langle 00 \rangle$, $\chi_2 = \langle 55 \rangle$, $\chi_3 = \langle FF \rangle$, $\chi_4 = \langle AA \rangle$.
4. $\alpha = \langle FF \rangle$, $\beta = \langle FF \rangle$, $\gamma = \langle FF \rangle$, $\delta = \langle FF \rangle$,
 $\chi_1 = \langle FF \rangle$, $\chi_2 = \langle FF \rangle$, $\chi_3 = \langle FF \rangle$, $\chi_4 = \langle FF \rangle$.

The data for the loop iterations is tabulated above. The program flow drops through after four loop executions. The next few instructions test the flag generation, before changing to the other synchronous modes to test their connectivity.

| Strobe In | Input Data | | Half-Port Registers | | | | Program Instructions |
|-----------|------------|--------|---------------------|--------|--------|--------|----------------------|
| | Port 1 | Port 0 | Reg 0H | Reg 0L | Reg 0H | Reg 0L | |
| | | | < FF > | < FF > | < FF > | < FF > | LI OR < 0000 > |
| | | | | | | | OP NULL except WOR0 |
| | | | | | | | LI CM=< 0000 > |
| $\phi 2$ | < 00 > | < 00 > | | | | | OP TEST IN.INIT |
| $\phi 3$ | < 00 > | < 00 > | | | | | |
| | | | < FF > | < FF > | < FF > | < FF > | OP WDV RIN1 WOR0 |
| $\phi 2$ | < 00 > | < FF > | | | | | OP IN.INIT RIN0 WOR0 |
| $\phi 3$ | < AA > | < 55 > | | | | | |
| | | | < 00 > | < AA > | < FF > | < 55 > | LI IPP IO M5 |
| $\phi 2$ | < FF > | < AA > | | | | | OP RIN01 WOR0 |
| $\phi 3$ | < 55 > | < 00 > | | | | | IN.INIT |
| | | | < 00 > | < FF > | < AA > | < 55 > | OP RIN1 WOR0 |
| | | | | | | | LI IPP IO M7 |

| | | |
|----------|---------------|----------------------|
| $\phi 2$ | < FF > < 55 > | OP IN.INIT RIN0 WOR0 |
| $\phi 3$ | < 00 > < AA > | |

< 00 > < FF > < 55 > < AA > OP RIN1 WOR0

This concludes the testing of the synchronous states, except for the reading out of the final input value resulting from the last instruction above. This is merged into the first few instructions of the asynchronous mode tests, the coding for which appears below (the two parts are executed consecutively without a pause between them.)

| Strobe In | Input Data | | Half-Port Registers | | | | Program Instructions |
|-----------|------------|--------|---------------------|--------|--------|--------|----------------------|
| | Port 1 | Port 0 | Reg 0H | Reg 0L | Reg 0H | Reg 0L | |

< 00 > < FF > < 55 > < AA > LI IPP M1

< F0 > < A5 > < 00 > < FF > < 55 > < AA > OP IN.INIT RIN0 WOR0

< F0 > < A5 > OP RIN0 WOR0

| | | |
|----------|---------------|--------------|
| $\phi 3$ | < A5 > < F0 > | OP RIN1 WOR0 |
|----------|---------------|--------------|

| | | |
|----------|---|--------------|
| $\phi 1$ | < A5 > < A5 > < 00 > < FF > < F0 > < AA > | OP RIN0 WOR0 |
|----------|---|--------------|

OP IN.INIT RIN1 WOR0

LI IPP IO M0

$\phi 3$ < 00 > < AA > < 00 > < FF > < 55 > < AA > OP IN.INIT RIN0 WOR0

$\phi 1$ < FF > < 55 > < 00 > < 00 > < AA > < AA > OP RIN0 WOR0

$\phi 3$ < FF > < 55 > OP IN.INIT RIN1 WOR0

$\phi 1$ < FF > < 55 > < 00 > < FF > < 55 > < AA > OP WDV RIN0 WOR0

LI IPP IO M2

$\phi 3$ < 5A > < 0F > < 00 > < FF > < 55 > < AA > OP IN.INIT RIN1 WOR0

< 5A > < F0 > < 00 > < FF > < 0F > < AA > OP RIN0 WOR0

< 5A > < F0 > OP RIN1 WOR0

< 5A > < F0 > OP RIN0 WOR0

< 5A > < F0 > OP WVD RIN0 WOR0
idle

$\phi 3$ < 5AF0 > idle

< 00 > < FF > < 0F > < F0 > execute

$\phi 3$ < AA > < 55 > OP RIN1 WOR0

| | | |
|----------|---|--------------------------|
| | | OP RIN0 WOR0 |
| | | LI IPP IO M4 |
| $\phi 3$ | < 5A > < A5 > < 00 > < FF > < 0F > < F0 > | OP IN.INIT RIN1 WOR0 |
| | < A5 > < 00 > < 5A > < FF > < 0F > < F0 > | OP RIN1 WOR0 |
| | < A5 > < 00 > < 5A > < FF > < 0F > < F0 > | OP RIN0 WOR0 |
| | < A5 > < 00 > < 5A > < FF > < 0F > < F0 > | OP RIN1 WOR0 |
| | < A5 > < 00 > < 5A > < FF > < 0F > < F0 > | OP RIN1 WOR0 |
| | < A5 > < 00 > < 5A > < FF > < 0F > < F0 > | OP WDV RIN1 WOR0 idle |
| $\phi 3$ | < A5 > < 00 > | idle |
| | < 5A > < A5 > < 0F > < F0 > | execute |
| $\phi 3$ | < 00 > < FF > | OP RIN0 WOR0 |
| | | OP RIN1 WOR0 |
| | | OP RIN0 WOR0 |

This completes the test program for the Input Processor. Due to the

predominance of straight line coding in this test sequence, the load time of 4020 cycles for the 60 instructions required far outweighs the total execution time of 80 clock cycles. The total test length of 4100 clock cycles takes about 0.51 milliseconds at design clock speed.

A.6 The Data Memories.

This section of test is designed to be run as sequences of instructions run under program control by the Execution Controller. To improve test times, where identical instruction sequences are required together with a few different instructions, use is made of the ability to Write to any particular address in the CS, to allow changes to be made to those instructions requiring it while keeping all others the same. Also, obviously, tests can be applied to all DMs in parallel, although the resultant data must be taken off-chip via the Output Processor one word at a time.

The allowable instructions to a DM are:

RP n Im Op or LI: Read, pointer $n = 0, 1$, increment $m = 0, 1$, *preset* :
 $-16 \leq \textit{preset} \leq 15$.

WP n Im Op or LI: Write, pointer $n = 0, 1$, increment $m = 0, 1$, *preset* :
 $-16 \leq \textit{preset} \leq 15$.

LI P n C LI Pointer n Control (Address, Increment), $n = 0, 1$, *both*.

NULL No operation.

@ n Jump address n .

First, to check the buffers for API faults, LI data words consisting of API patterns to the buffers. Note that since the cells of the data buffers are interleaved (A0,B0,A1,B1,..etc.), the API patterns are not identical to those applied to a simple register. The notation $RPnImx4$ is taken to mean that four similar instructions are applied, one to each of the four DMs in turn, with the Ringbus appropriately switched in each case to allow the resultant data to be latched by the Output Processor, and subsequently output. The instruction sequence loaded is:

LIP0C Ad= $\langle 00 \rangle$

LIP1C Ad= $\langle 1F \rangle$

LI WP0I0 LI Data= $\langle XXXX \rangle$

LI WP1I0 LI Data= $\langle YYYY \rangle$

RP0I0x4

RP1I0x4

@1 LI JPC @1 (*Halt*)

This sequence is loaded with the initial values of $\langle XXXX \rangle$ and $\langle YYYY \rangle$, executed and then only the two Write instructions are changed by scanning two Write instructions in with new values for $\langle XXXX \rangle$ and $\langle YYYY \rangle$. Four pairs of values are used for the complete sequence:

1. $\langle XXXX \rangle = \langle 5555 \rangle$, $\langle YYYY \rangle = \langle 5555 \rangle$

2. $\langle XXXX \rangle = \langle 5555 \rangle$, $\langle YYYY \rangle = \langle AAAA \rangle$

3. $\langle XXXX \rangle = \langle AAAA \rangle$, $\langle YYYY \rangle = \langle 5555 \rangle$

4. $\langle XXXX \rangle = \langle AAAA \rangle$, $\langle YYYY \rangle = \langle AAAA \rangle$

To test for column faults, simply re-read the buffers after the final API test case without rewriting them: this allows the values read from memory to

the buffers after the previous Reads to be output, and this can be checked for SA1 and SA0 faults. This is simply accomplished by resetting the IPC to the address of the first Read. The next portion of the test is the cell test. Direct addressing must be used, via the LI PnC instructions. First the array must be initialised.

```
LI PBC Ad=< N >  
LI WP0I0 < 0000 >  
@1 LI JPC @1
```

By loading this segment, executing it, and repeating the load/execute until N has been cycled from 0 to 32, the array is initialised to all zeros. The actual test consists of iterations of reading a word, writing it, reading it again to verify the write, checking the word below for any effects of the write, and then reading the word above, starting the next iteration. This is accomplished with the following loaded segment.

```
LI BPC Ad=< N >  
RP0I0x4  
LI WP0I0 < DDDD >  
RP0I0x4  
LI PBC Ad=< N - 1 >  
RP0I0x4  
@1 LI JPC @1
```

Each reload only requires two changed instructions. N is cycled from 0 to 32 with $\langle DDDD \rangle = \langle FFFF \rangle$. Then the program segment is completely rewritten, keeping the same form but substituting pointer 1 for pointer 0. The cycle is repeated, with $\langle DDDD \rangle = \langle 0000 \rangle$ marching zeros up the array. This test, as well as verifying the cells, verifies the

uniqueness of the decoding of each pointer. It is required that a correlation between the two be established. This is done by loading a segment that writes a unique word to each address of one pointer, and checks that the same address in the other pointer fetches that unique word.

```
LI BPC Ad=< N >  
LI WP0I0 < ZXZP >  
OP RP0I0all (flush the buffers)  
RP0I0x4  
RP1I0x4  
@1 LI JPC @1
```

The "unique word" $\langle ZXZP \rangle$ could be such that $\langle Z \rangle$ is the five-bit address of the word, $\langle X \rangle$ is the complement of $\langle Z \rangle$, and $\langle P \rangle$ is a single parity bit. It is now used as a flag, to determine if the auto-increment circuitry selects the correct address, and by inference to indicate its correct operation. The increment circuit is tested first, with a simple loop testing it exhaustively.

```
LI CM=(LOOP0)  
LI LOOP0=32  
@1 RP0I1x4  
RP1I1x4  
JNZ LOOP0 @1  
@2 LI JPC @2
```

The increment loop tests a large part of the allowable states of the auto-increment adder. It remains to test the "two-sided" operations, and this is best done using a test set that pseudo-exhaustively tests the adder cells. The inputs can be set by a LI P n C command that alters both current

address and pointer preset increment, and a $RPnIP$ command will execute the addition. A further read is required to access the flag data pulled down by the lookahead Read.

```
LI BPC Ad=< N > P=< p >
OP RP0IPall (add, flush)
OP RP1IPall
RP0I0x4 (flag out)
RP1I0x4
@1 LI JPC @1
```

This sequence needs to be applied for nine pairs of values of $\langle N \rangle$ and $\langle p \rangle$, listed below (written as five-bit binary patterns, MSB at left, LSB at right). This completes the test of the pointer auto-incrementing circuits, and of the DMs as a whole.

1. Address $\langle N \rangle = (00010)$, Increment $\langle p \rangle = (11100)$
2. Address $\langle N \rangle = (00010)$, Increment $\langle p \rangle = (11101)$
3. Address $\langle N \rangle = (00011)$, Increment $\langle p \rangle = (11100)$
4. Address $\langle N \rangle = (00011)$, Increment $\langle p \rangle = (11101)$
5. Address $\langle N \rangle = (00000)$, Increment $\langle p \rangle = (11111)$
6. Address $\langle N \rangle = (01010)$, Increment $\langle p \rangle = (01010)$
7. Address $\langle N \rangle = (10100)$, Increment $\langle p \rangle = (10100)$
8. Address $\langle N \rangle = (11111)$, Increment $\langle p \rangle = (11111)$

9. Address $\langle N \rangle = (00000)$, Increment $\langle p \rangle = (11110)$

The test length of the DM test, in clock cycles, may be broken down as follows:

Data Buffers, Column Faults:

Load = 5092 Execute = 424

Cell Test Initialisation:

Load = 2278 Execute = 96

Cell Test:

Load = 10452 Execute = 1024

Decoder Correlation:

Load = 4958 Execute = 384

Incrementer Test:

Load = 804 Execute = 300

Pseudo-Exhaustive Adder Test:

Load = 1340 Execute = 108

Total Test Cycles = 27260

At the design clock rate of 8 MHz, the complete DM test will take about 3.4 milliseconds.

A.7 The Adder/Subtractor/Shifter/ Accumulators.

The Adder/Subtractor/Shifter/Accumulators (ASSAs) are tested as functional units by applying a test program consisting of subset of the ALU instruction set, together with appropriate data. The tests are designed to find any adder cell faults, any SA control lines and any SA or API faults in the data paths and registers. Each ASSA can be thought of as consisting of four main partitions, those being the shifter/accumulator, the adder, and the two separate input stages. The shifter/accumulator can be verified independently of all other partitions, except with regard to its interface with the adder. The adder and the input stages are all too closely connected to allow full fault location for every possible fault, but by structuring the test with regard for the circuit details of the partitions, and by taking a pseudo-exhaustive approach wherever possible, most faults will be attributable to a specific circuit element or area.

Generally tests are applied in parallel to all four ASSAs, with the outputs taken to the pins via the Output Processor on eight consecutive clock cycles (4 ASSAs with two 16 bit accumulator halves each). The exceptions occur when the output of one accumulator is used as the input to another, and where the flag generation and returns to the PSR are being tested. In the test specification below it is assumed that the instructions are meant to apply to all of the modules unless specifically noted otherwise. It is assumed that a DM has been filled with the four values of the API test set (< *AAAA* >, < *5555* >, < *0000* >, < *FFFF* >) prior to the test. Also

the Output Processor is assumed to be preset to a synchronous dual 8 bit byte transfer mode.

The notation does not include path specifications around the Ringbus: generally the connectivity is self-evident from consideration of the position of the modules actually reading or writing during that cycle. Another notational peculiarity arises because every accumulator read is required to be latched by the Output Processor and output, and hence this latching is assumed and not specifically noted every time. Also the sequences of instructions reading the high or low bytes from all four modules in turn and passing the values to the pins via the Output Processor are represented by single "pseudo-instructions" **HWRACCx4** or **LWRACCx4**. The following list expands and explains the mnemonics used in this section.

HWRACC_n Write to the Accumulators high bytes, all or $n =$ named one(s).

LWRACC_n Write to the Accumulators low bytes, all or $n =$ named one(s).

BWRACC_n Write to the Accumulators both bytes, all or $n =$ named one(s).

BRDACC_n Read from the Accumulators both bytes, all or $n =$ named one(s).

HRDACC_n Read from the Accumulators high bytes, all or $n =$ named one(s).

LRDACC_n Read from the Accumulators low bytes, all or $n =$ named one(s).

LRDACCx4 Read in turn from each Accumulator, low bytes to output.

HRDACCx4 Read in turn from each Accumulator, high bytes to output.

SHRACC_n Shift Accumulator right one bit, all or $n =$ named one(s).

SHLACC_n Shift Accumulator left one bit, all or $n =$ named one(s).

CLRACC_n Clear Accumulator, all or $n =$ named one(s).

NEGACC_n Negate Accumulator, all or $n =$ named one(s).

SUBACC_n Subtract bus from Accumulator, all or $n =$ named one(s).

NEGBUS_n Negate bus, store in Accumulator, all or $n =$ named one(s).

INCACC_n Increment Accumulator, all or $n =$ named one(s).

ADDACC_n Add bus to Accumulator, all or $n =$ named one(s).

ADDAC2_n Output accumulator n to bus to add into adjacent accumulator.

@ N Jump address N .

LID Load Immediate Destination.

BALAC LID; Both bytes All Accumulators.

The first section of code below tests the accumulators for API and SA input and output faults, for correct ($ACC < 0$) flag generation, and for correct shift left and shift right operations.

OP HWRACC < AAAA > from DM
HRDACCx4
LRDACCx4
OP LWRACC < AAAA > from DM
HRDACCx4
LRDACCx4
LI CM (ACC A < 0)
TEST LI CM (ACC B < 0) (*check NOP for result*)
TEST LI CM (ACC C < 0)
TEST LI CM (ACC D < 0)
TEST LI CM (ACC A < 0)
OP HWRACC < 5555 > from DM
HRDACCx4
LRDACCx4
OP LWRACC < 5555 > from DM
HRDACCx4
LRDACCx4
TEST LI JPC @1 (*check NOP for result*)
LI CM (ACC B < 0)
TEST LI JPC @1
LI CM (ACC C < 0)
TEST LI JPC @1
LI CM (ACC D < 0)
TEST LI JPC @1
@1 OP SHLACC
HRDACCx4
LRDACCx4
OP SHRACC
HRDACCx4
LRDACCx4

OP HWRACC < 5555 > from DM
 OP SHRACC
 HRDACCx4
 LRDACCx4
 OP HWRACC < AAAA > from DM
 OP SHLACC
 HRDACCx4
 LRDACCx4
 OP BWRACC < 0000 > (*use sign extend*)
 HRDACCx4
 LRDACCx4
 OP SHRACC
 HRDACCx4
 LRDACCx4
 OP SHLACC
 HRDACCx4
 LRDACCx4
 LI < FFFF > LID=BALAC (*test LI lines*)
 HRDACCx4
 LRDACCx4
 ©2 LI JPC ©2

This sequence is essentially all straight line coded, and its size is limited by the available memory space of 128 words, and the ease of restarting the test sequence after a memory reload. The next block to be loaded continues with some straight line code to complete the load and shift tests, and then uses loops to shift a single bit shift across each of the accumulators in turn, checking each bit position for its effect on the (ACC=0) flag. The remainder of the block begins applying test patterns to the adder and input stages,

completing all but one of the tests possible with patterns of all zeros and all ones.

```
LI BWRACC < FFFF >
OP SHRACC
HRDACCx4
LRDACCx4
OP SHLACC
HRDACCx4
LRDACCx4
OP SHRACC
OP CLRACC
HRDACCx4
LRDACCx4
LI LWRACC < 0001 >
LI CM (ACC A = 0)
@1 TEST LI JPC @2
SHLACC A
LI JPC @1
@2 LI CM (ACC B = 0)
@3 TEST LI JPC @4
SHLACC B
LI JPC @3
@4 LI CM (ACC C = 0)
@5 TEST LI JPC @6
SHLACC C
LI JPC @5
@6 LI CM (ACC D = 0)
@7 TEST LI JPC @8
SHLACC D
```

LI JPC @7
@8 OP ADDACC < 0000 >
HRDACCx4
LRDACCx4
OP INCACC
HRDACCx4
LRDACCx4
OP CLRACC
OP ADDACC < FFFF >
HRDACCx4
LRDACCx4
OP ADDACC < 0000 >
HRDACCx4
LRDACCx4
OP INCACC
HRDACCx4
LRDACCx4
OP NEGBUS < 0000 >
HRDACCx4
LRDACCx4
OP SUBACC < FFFF >
HRDACCx4
LRDACCx4
OP BWRACC < FFFF >
OP NEGACC
HRDACCx4
LRDACCx4
@9 LI JPC @9

The last all zero pattern test is carried over to the third block of tests due to lack of space. After this, 32 bit patterns of alternating ones and zeros are assembled in the accumulators, and manipulated and operated on to further test the adder. As identical tests and operations are carried out using the < **AAAA** > and < 5555 > patterns, a loop is used to repeat the sequence, and the auto-incrementing DM pointer is used to fetch the correct data value for successive loop iterations. One further instruction completes the single-bit adder cell tests, and there remains only the generation tests for the carry lookahead. These require patterns not commonly used in the chip test, so they are input from the pins on the cue of the IREQ pin. As the two pairs of adjacent ASSAs must cooperate to allow testing, a test sequence can be constructed to maximise the usage of data and minimise the instructions loaded by using a pair of values to test one parallel pair of ASSAs and then using the same data in reversed positions to test the other parallel ASSA pair. The data set that must be applied is tabulated below, after the code sequence.

```

OP CLRACC
OP NEGACC
HRDACCx4
LRDACCx4
LI CM (LOOP0≠0)
LI LOOP0=1
@1 OP LWRACC < data > RP0I0 DM
OP HWRACC < data > RP0I0
OP ADDACC < 0000 >
HRDACCx4
LRDACCx4
OP NEGACC
HRDACCx4

```

LRDACCx4
 OP CLRACC A,C LWRACC B,D RP0I0 DM
 OP NULL A,C HWRACC B,D RP0I1 DM (*fetch next value*)
 OP ADDACC A,C ADDAC2 B,D
 HRDACCx2 A,C CLRACC B,D
 LRDACCx2 A,C NULLx3 B,D
 OP ADDAC2 A,C ADDACC B,D
 HRDACCx2 B,D CLRACC A,C
 LRDACCx2 B,D NULLx3 A,C
 OP SUBACC A,C SUBAC2 B,D
 OP HRDACC A
 OP LRDACC A
 OP HRDACC C BRDACC B BWRACC A
 OP LRDACC C CLRACC B NULL A
 OP SUBACC B SUBAC2A BWRACC C BRDACC D
 OP HRDACC B CLRACC D NULL A,C
 OP LRDACC B SUBACC D SUBAC2 C NULL A
 OP HRDACC D BWRACC B BRDACC A NULL C
 OP LRDACC D ADDACC A ADDAC2 B NULL C
 OP HRDACC A BRDACC C BWRACC D NULL B
 OP LRDACC A ADDACC C ADDAC2 D NULL B
 OP HRDACC C BWRACC A BRDACC B NULL C
 OP LRDACC C ADDACC B ADDAC2 A NULL C
 OP HRDACC B BWRACC C BRDACC D NULL A
 OP LRDACC B ADDACC D ADDAC2 C NULL A
 OP HRDACC D NULL A,B,C
 OP LRDACC D NULL A,B,C
 TEST LI JPC @1 (*jump not zero*)
 LI LID=BALAC < *FFFF* >
 OP SUBACC < 0000 > from DM

```

HRDACCx4
LRDACCx4 IN.INIT (fetch first data for loop)
LI LOOP0=7
@2 OP HWRACC B,D < data1 > from IPP
OP LWRACC B,D < data1 > from IPP IN.INIT
OP HWRACC A,C < data2 > from IPP
OP LWRACC A,C < data2 > from IPP
OP ADDAC A,C ADDAC2 B,D
OP HRDACCx2 A,C
OP LRDACCx2 A,C
OP HWRACC A,C < data2 > from IPP
OP LWRACC A,C < data2 > from IPP IN.INIT
OP ADDACC B,D ADDAC2 A,C
OP HRDACCx2 B,D
OP LRDACCx2 B,D
LI TEST JPC @2 (jump not zero)
@3 LI JPC @3

```

The values < data1 > and < data2 > are listed below.

1. < data1 > = < 0101 > < data2 > = < 0F0F >
2. < data1 > = < 0F0F > < data2 > = < 0101 >
3. < data1 > = < 1010 > < data2 > = < FOFO >
4. < data1 > = < F0F0 > < data2 > = < 1010 >
5. < data1 > = < 2222 > < data2 > = < EEEE >
6. < data1 > = < EEEE > < data2 > = < 2222 >
7. < data1 > = < 4444 > < data2 > = < CCCC >

8. $\langle data1 \rangle = \langle CCCC \rangle$ $\langle data2 \rangle = \langle 4444 \rangle$

This completes the test of the ASSAs. Three program blocks are loaded and executed, consisting in total of 336 instructions, taking 22502 cycles to load and 981 cycles to execute. The total test length is 23483 clock cycles, or about 2.94 milliseconds at design clock speed.

A.8 The Multiplier/Dividers.

This section of test is organised as six subsections, each exercising different aspects of the modules. Of those six, two are straight line coded to perform once off testing of the flag conditions, while the rest are coded as fixed iteration loops, controlled by the Execution Controller. The data required for the tests is fetched from the DM (four API patterns, assumed to be preloaded during preceding tests) and from the Input Processor (other data sets). The Input Processor must be loaded with a synchronous I/O mode, but the one used may be selected by the tester. The sections are intended to be concatenated into one loadable and executable program. Except where noted otherwise, all instructions are to apply identically to each of the four multiplier/dividers, as this test exercises them in parallel, only taking the outputs out separately. Outputs are managed by setting the Output Processor to a synchronous mode and tying the OREQ pin high.

As well as those mentioned in previous sections, the following abbreviations are used to allow compact notation:

LI LOOP n = p LI Loopcountern with (decimal integer) p .

JNZ LOOP n LI TEST JPC with CM=(Loopcountern non-zero flag).

WAIT x WAIT "Muldiv x Complete" flag.

MVDDGO ALU instruction: move dividend from Acc. to Div., start division.

LODDGO ALU instruction: load dividend from bus to Mult., start division.

TEST-MVDDGO TEST version of MVDDGO, one adder op. only.

GOX Start multiplication (numbers loaded previously).

TEST-LDGOX TEST version of LDGOX.

LDGOX ALU instruction: load multiplicand from bus, start multiply.

IN.INIT Input Initiate microcode bit to Input Processor.

MOVAX Move 32 bit product register content to Accumulator.

TEST < 0000 > Test conditional evaluation, with CM=< 0000 >.

HWRACC $x4$ Read accumulators' high bytes to output processor (one at a time) and force them to be output by tying OREQ high.

LWRACC $x4$ As for HWRACC $x4$, but low byte.

IPP Input Processor.

@ n Jump address n .

```

LI IO M5 (any synchronous mode)
LI P0 INCREMENT=8 (cycle through 4 API values)
OP IN.INIT (input DMI values as required)
LI LOOP0=3
LI CM (LOOP1)
@0 READ IPP, LATCH DMIs
LI LOOP1=3
@1 TEST-LDGOX RP0IP (read API value)
NULL (wait for end of op.: flag not verified.)
MOVAX
HRDACCx4
LRDACCx4
JNZ LOOP1 @1
LI CM (LOOP0)
JNZ LOOP0 @0

```

The data set required at the IPP pins (to latch in and pass to the DMI) consists of < 0001 >, then < 0000 >, < 0003 >, and finally < 0002 >. This detects most of the sensitised paths' API faults, and about half of the adder faults. The next section detects most of the remaining adder faults, using TEST division operations to perform additions with a non-zero accumulator. The dividend and divisor values read in via the IPP each loop are listed below the code.

```

LI LOOP0=12
@2 OP IN.INIT
LOADDV from IPP, IN.INIT
HWRACCs from IPP, IN.INIT
LWRACCs from IPP
TEST-MVDDGO

```

NULL
MOVAX
HRDACCx4
LRDACCx4
JNZ LOOP0 @2

1. Dividend = < 80000000 >, Divisor = < 0000 >
2. Dividend = < **AAAAAAAA** >, Divisor = < 0000 >
3. Dividend = < **7FFFFFFF** >, Divisor = < 8000 >
4. Dividend = < 55555555 >, Divisor = < 8000 >
5. Dividend = < 80000000 >, Divisor = < **FFFF** >
6. Dividend = < **FFFFFFFF** >, Divisor = < **FFFF** >
7. Dividend = < **7FFFFFFF** >, Divisor = < 0000 >
8. Dividend = < 55555555 >, Divisor = < **AAAA** >
9. Dividend = < **AAAAAAAA** >, Divisor = < 5555 >
10. Dividend = < 49248000 >, Divisor = < **FFFF** >
11. Dividend = < **3FFF**8000 >, Divisor = < 9249 >
12. Dividend = < **B6DB**0000 >, Divisor = < 2492 >
13. Dividend = < 92490000 >, Divisor = < **6D6B** >

The following straight-line coded sequence tests the flag operations under multiplication, and tests for correct cycle times in each multiplier. No input or output data examination is done, as the results are observable via the NOP external pin and the IREQ pin.

```
TEST < 0000 > GOX(A)
WAITA IN.INIT GOX(A)
WAITA IN.INIT
TEST < 0000 > GOX(B)
WAITB IN.INIT GOX(B)
WAITB IN.INIT
TEST < 0000 > GOX(C)
WAITC IN.INIT GOX(C)
WAITC IN.INIT
TEST < 0000 > GOX(D)
WAITD IN.INIT GOX(D)
WAITD IN.INIT
```

The fourth section tests those internal adder states which are not accessible using the single adder operation instructions, and those DMI shift and load capabilities not verified as yet. The coding is presented first, followed by a list of the operands input to the multipliers each loop execution via the IPP.

```
LI LOOP0=14
©3 LOAD DMI from DM IN.INIT
READ IPP LDGOX
WAITA MOVAX
HRDACCx4
LRDACCx4
JNZ LOOP0 ©3
```

1. Parallel Input = $\langle 0C0C \rangle$, DMI = $\langle 0001 \rangle$.
2. Parallel Input = $\langle 0AA0 \rangle$, DMI = $\langle 4000 \rangle$.
3. Parallel Input = $\langle 0AA0 \rangle$, DMI = $\langle 8000 \rangle$.
4. Parallel Input = $\langle 0AA0 \rangle$, DMI = $\langle C000 \rangle$.
5. Parallel Input = $\langle 4555 \rangle$, DMI = $\langle 0002 \rangle$.
6. Parallel Input = $\langle 6000 \rangle$, DMI = $\langle B000 \rangle$.
7. Parallel Input = $\langle 6FFF \rangle$, DMI = $\langle A000 \rangle$.
8. Parallel Input = $\langle F0F0 \rangle$, DMI = $\langle 0001 \rangle$.
9. Parallel Input = $\langle 080F \rangle$, DMI = $\langle 0007 \rangle$.
10. Parallel Input = $\langle F0F0 \rangle$, DMI = $\langle F000 \rangle$.
11. Parallel Input = $\langle B05A \rangle$, DMI = $\langle 0006 \rangle$.
12. Parallel Input = $\langle 70A5 \rangle$, DMI = $\langle 0005 \rangle$.
13. Parallel Input = $\langle FA50 \rangle$, DMI = $\langle FFFF \rangle$.
14. Parallel Input = $\langle FA50 \rangle$, DMI = $\langle AAAA \rangle$.
15. Parallel Input = $\langle FA50 \rangle$, DMI = $\langle 5555 \rangle$.

Division flag generation and cycle timing is tested by the following piece of straight line instruction coding. As previously, the results are available as flags on the NOP pin and the IREQ pin.

```

TEST < 0000 > LODDGO(A)
WAITA IN.INIT LODDGO(A)
WAITA IN.INIT
TEST < 0000 > LODDGO(B)
WAITB IN.INIT LODDGO(B)
WAITB IN.INIT
TEST < 0000 > LODDGO(C)
WAITC IN.INIT LODDGO(C)
WAITC IN.INIT
TEST < 0000 > LODDGO(D)
WAITD IN.INIT LODDGO(D)
WAITD IN.INIT

```

All that remains is to test the overflow decision and hard-limiting circuits. This is done by applying six division instructions, again with data from the external tester via the IPP. The last IN.INIT of the previous section loads the first DV value required. Dividends and divisors are listed below the code.

```

LI LOOP0=5
@4 LOADDV IN.INIT
HWRACCs IN.INIT
LWRACCs IN.INIT (for next iteration)
MVDDGO
WAITA MOVAX
HRDACCx4
LRDACCx4
JNZ LOOP0 @4

```

1. Dividend = < 08000000 >, Divisor = < F000 >

2. Dividend = $\langle 80000000 \rangle$, Divisor = $\langle FFFF \rangle$
3. Dividend = $\langle C0000000 \rangle$, Divisor = $\langle FFFF \rangle$
4. Dividend = $\langle 80000000 \rangle$, Divisor = $\langle 0001 \rangle$
5. Dividend = $\langle 3FFF0001 \rangle$, Divisor = $\langle 7FFF \rangle$
6. Dividend = $\langle 3FFFFFFF \rangle$, Divisor = $\langle 8000 \rangle$

The test length for this section of test breaks down as follows:

Load: 90 loaded instructions x 67 cycles = 6030 cycles

Block 1 Exe: $5 + 4 \times (4 + (4 \times 12)) = 213$ cycles

Block 2 Exe: $1 + (13 \times 16) = 209$ cycles.

Block 3 Exe: $4 \times 11 = 44$ cycles.

Block 4 Exe: $1 + (15 \times 20) = 301$ cycles.

Block 5 Exe: $4 \times 20 = 80$ cycles.

Block 6 Exe: $1 + (6 \times 31) = 187$ cycles.

Total: 6030 load + 1034 execute = 7064 cycles.

At the design clock speed, this would require about 0.88 milliseconds to complete.

A.9 Total Test Length.

The lengths of the load/unload, execute and combined test sequences are tabulated below. The complete chip test requires a total of 210157 clock cycles to carry out, and this will take about 26 milliseconds at the design clock rate of 8 MHz.

| Partition | Load Cycles | Run Cycles | Total Cycles |
|------------------|-------------|------------|--------------|
| Control Core | 126272 | 5542 | 131814 |
| ALU/DM Decoders | 4600 | 120 | 4720 |
| Output Processor | 2345 | 55 | 2400 |
| Ring Bus | 9179 | 137 | 9316 |
| Input Processor | 4020 | 80 | 4100 |
| Data Memories | 24924 | 2336 | 27260 |
| ASSAs | 22502 | 981 | 23483 |
| Multiplier/Div's | 6030 | 1034 | 7064 |
| Totals | 199872 | 10285 | 210157 |

Appendix B

Multi-Project Chip Test Partitions.

The multi-project chip (MPC) concept [94,95] allows several different circuits to be merged into a single die for fabrication, with each of the circuits having separate bonding pads for the input and output of signals. In packaging, a circuit is selected from those on the fabricated die by selectively bonding the particular circuit's I/O pads. The rationale for this procedure is that it allows small fabrication runs of the chips while distributing the costs of the processing among a number of clients. This allows the cost of prototyping and verifying circuits to be substantially reduced, thereby allowing participation in custom digital circuit design by a greater number of organisations and institutions, who might otherwise be deterred by the high initial costs.

When a particular manufacturer or intermediary organisation assembles

a MPC, the individual circuit designs are usually restricted with regards to the overall size and the maximum number of pins allowed to be allocated. For instance, the CSIRO coordinated the AUSMPC series of MPCs, and the circuit size limitations were set at 40 I/O pins, and an absolute maximum size of 6627 x 7030 microns (or at $\lambda = 2.5$ microns, 2812 x 2650 λ). The size quoted is the absolute maximum, and generally the smaller the configuration, the better chance there is of being able to merge it with other circuits and gain the cost benefits.

Bearing these restrictions in mind, the following sections suggest particular configurations of the major TFB partitions to facilitate their fabrication as MPCs.

B.1 The Data Memory and Decoder.

The Data Memory (DM) and the Decoder which converts the DM's instruction microcode field into control lines may readily be joined to form a single MPC, as illustrated in Figure B.1. The Decoder requires a 4 bit input field, as well as the global NOP line and the LI/Op instruction bit, and since a scan path is incorporated in its output latches, it requires only a further 3 pins to ensure its testability, those being Scan In, Scan Out and Scan Clock. This scan path also allows the testing the body of the DM in the case of a Decoder failure. The DM itself communicates solely via its 16 bus connections. A further 2 pins are required for the voltage rails, and another 3 for the clock phases, bringing the total pin count to 30.

As the actual layout of the Decoder awaits finalisation, the complete

module cannot be sized. The DM, including all of the pointer control circuits, is dimensioned at about 2500 lambda by 2200 lambda. If this configuration is too large for a particular MPC specification, given the value of lambda for the process, then size reductions may be made by reducing the number of columns of RAM implemented, with a commensurate decrease in the numbers of associated data buffers. This is preferable to the omission of the decoder, as it is of value to test the two parts working together, to get an idea of allowable clock timing. The memory should not be reduced below 8 columns width, as the resultant configuration would not test the memory address line buffers, which are inserted after every four cells of a memory row.

B.2 The Input and Output Processors.

Heuristically these two partitions are obvious candidates to share an MPC die, as they may then be used to test the implementation of the communications protocol. However, the high pinout that each requires in their functional configurations is beyond the scope of most MPC specifications, so a reduced configuration is proposed in Figure B.2. Under this proposal, each half-port and register half is reduced to a single bitslice, giving a minimal configuration. The rest of the circuitry is implemented as for the functional case, with only 8 bits of the data bus being required. The NOP and LI I/O Control lines for both circuits may be commoned at pins, as may the clocks and the voltage rails. The total number of pins required for the single bitslice register configuration proposed is 32: 8 bus, 2 in, and

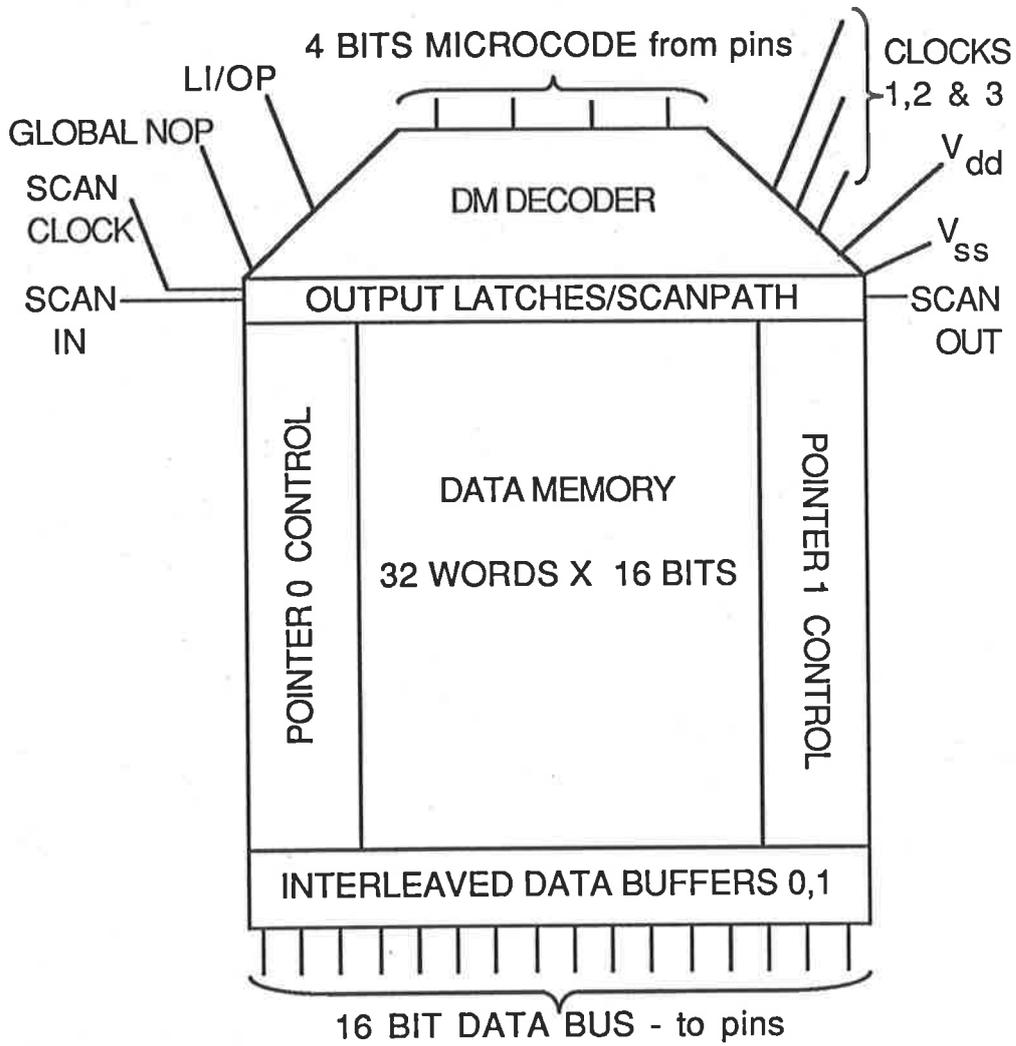


Figure B.1: Data Memory MPC Configuration.

2 out pins for the data transfer, 4 pins for communications handshaking (Strobe In and Out, IREQ and OREQ), 7 common pins (NOP, LI I/O Control, Clock Phases 1,2 and 3, V_{DD} and V_{SS}), 3 input and 2 output instruction microcode bits, 2 status flags, and 2 LI Regn lines to the Output Processor. If one spare pin and space are available, then this is a good configuration in which to verify the characteristics of the Pass/Break switches, by adding a four bit switch array between the two halves of the bus in the position indicated by the broken outline: if they work correctly then they do not affect the bus when switched open, and similarly if they fail in the open state, while if they fail and short bus segments together, they may be removed from circuit by the careful use of a scalpel, particularly if the routing is deliberately left long and exposed. Alternatively, if the links from the bus to the switches include two pads suitable for microprobing, then by applying a high current between the two microprobe pads the links may individually be fused and blown open.

B.3 The ALU.

The ALU, consisting of a multiplier/divider and an adder/subtractor/-shifter/accumulator, requires some 25 control lines from the Decoder, which decodes a 5 bit instruction field to generate them. While the fullsize ALU is rather large (approximately 3000 lambda wide by 3000 lambda high), it is not feasible to implement it without a decoder to reduce the input pin count. The multiplier also requires a Delocated Multiplier Input register to apply the second operand, and to apply the test set designed for the

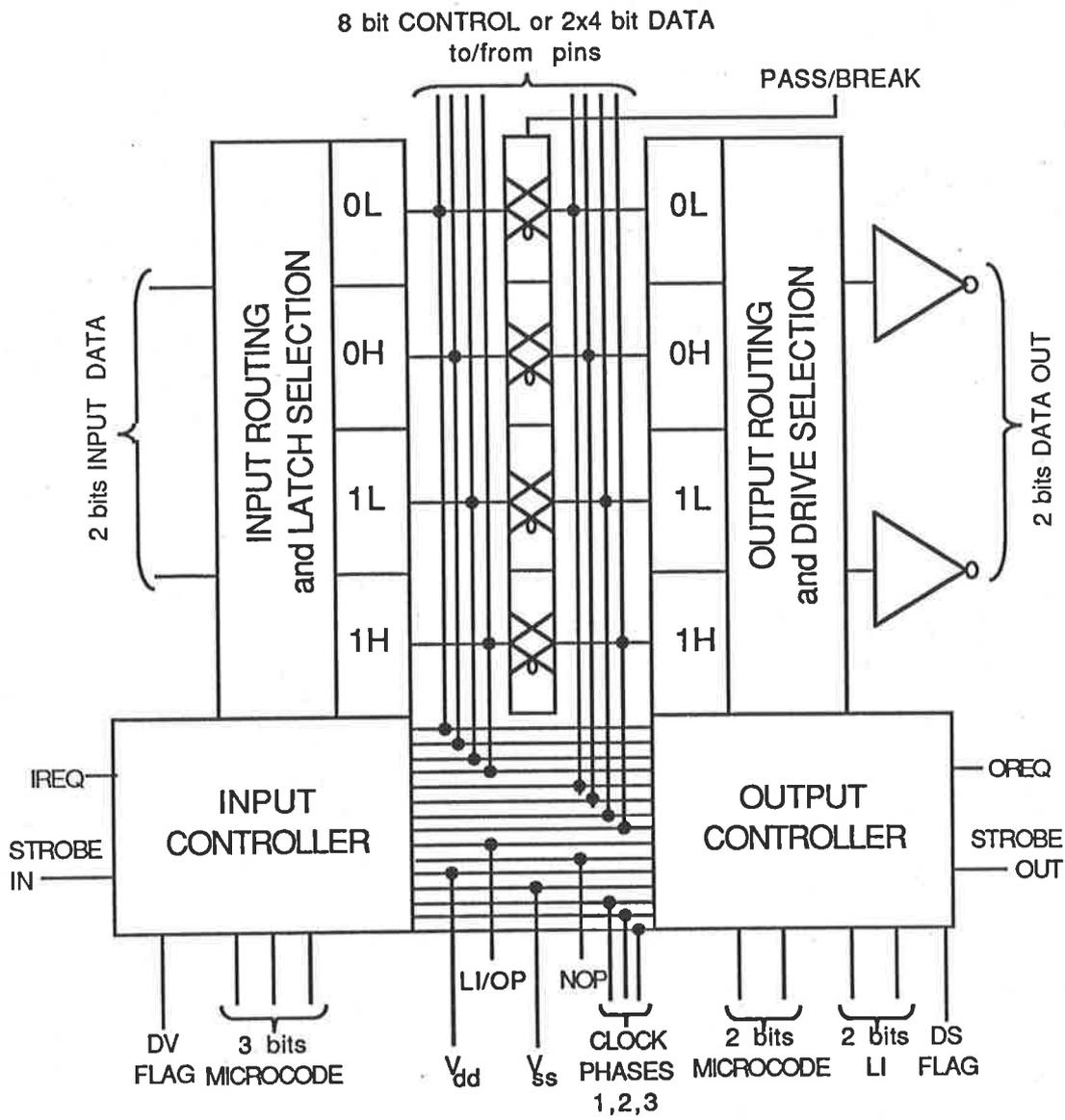


Figure B.2: I/O Processors' MPC Configuration.

multiplier (see Appendix A) this DMI must contain a different multiplier-cand to that in the main parallel input register. This may be arranged by interposing a T-switch, which also cuts the bus pin requirements from 32 to 16 pins, at the cost of 2 extra control pins and a small amount of active area. This arrangement is illustrated in Figure B.3. Note that the ALU Decoder output latches form a scan path, which requires three extra pins to operate. The total pin count is 36 pins: 16 bits of bus, 5 bits ALU instruction field, 3 scan path lines, 3 clock lines, 2 voltage rails, 2 LI Accumulator Half lines, 2 instruction bits for the T-switch, and 1 bit each for the DMI latch, NOP and LI/Op. This configuration may be impossible to fabricate as an MPC for reasons of size: for instance, on a five micron feature size process ($\lambda = 2.5$ micron) such as that of Amalgamated Wireless (Australasia) Ltd. (AWA), the configuration would cover more than the 6 millimetre square usually allocated for MPCs (a cost criterion, not a technical consideration). This may be overcome by reducing the number of parallel bitslices in the various modules, but this is not as trivial as in the case of the I/O processors, for instance, as the multiplier/divider control circuits are designed to operate for a specific number of bits, and so must also be altered to account for the reduction in datapath bitslices.

B.4 The Control Core Elements.

The biggest element of the control core is the Control Store (CS), an 8kbit RAM array: far too large to be an MPC circuit. It may be considered useful to fabricate a single fullsize column and row, possibly physically

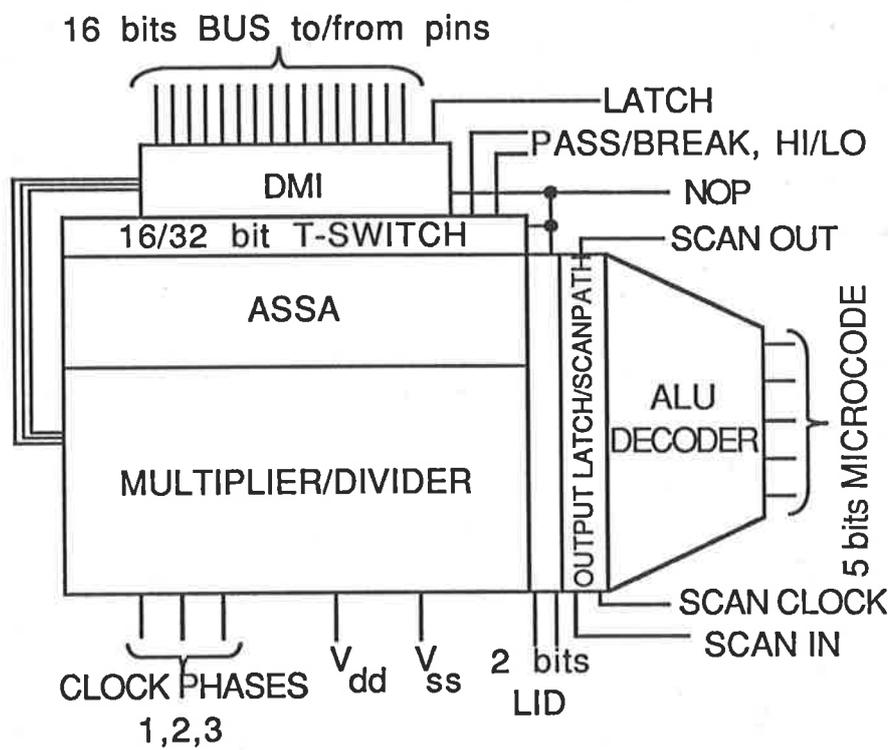


Figure B.3: ALU MPC Configuration.

reorganised into a more convenient shape, to investigate the circuit delays in the memory, but as the process used for the MPC is likely to be different to the one used for the final chip fabrication, attempts to extrapolate the results to the full production array are likely to be wildly inaccurate.

One approach to testing the control core would be to implement only the four columns of RAM actually required for program control, but this still gives too large a module: roughly 3000 transistors for the RAM cells alone, without even considering the decoders, the pointers, the BARs, the Write Register, the Instruction Register and the column-switching multiplexors and latches (about 2000 transistors), or the wordline latches (about 3000 transistors), let alone the main Controller and all the ancillary circuits. In fact the reduced CS configuration alone may be too large to be feasible as an MPC circuit, and there is no further useful reduction in size that can be made without dismantling this configuration into separate modules.

It seems likely that, as the various ancillary modules of the control core are laid out, they too will have to be fabricated out of context: not as part of a system, but interacting at most with one or two adjacent modules fabricated on the same die. This is unsatisfactory in the general case, although it will suffice for elements such as the loopcounters, and the LID Decoder.

The minimal system which could usefully verify a major portion of the control core design would consist of the Execution controller, the conditional evaluation circuitry (PSR, WAIT decoder, CM, and comparator), and the two pointers IPC and JPC, with the inputs being latched from a 20 bit section of the IR. Even this will result in a reasonably large cir-

cuit: the conditional evaluation circuitry on its own is of the order of 1000 transistors, the pointers together are about 500 transistors including the incrementer, and the IR segment adds another 500 transistors, while the Execution Controller could be of the order of 1000 to 2000 transistors. This system, illustrated in Figure B.4, is a subset of the complete control core, with the same connectivities except where inputs from the pins are substituted for the CS column mux outputs, for the flag returns to the PSR (not all implemented) and for the LI JPC line, and where pins output the Decrement Loopcounter signals (instead of routing them to non-existent Loopcounters) and the IPC and JPC pointer selects (as the effects of different selects are not otherwise distinguishable). The total pin count for this configuration depends on the number and method of implementing the flag returns: the size of the conditional evaluation circuitry could be substantially reduced by implementing only a few pin-driven flag returns and their associated registers.

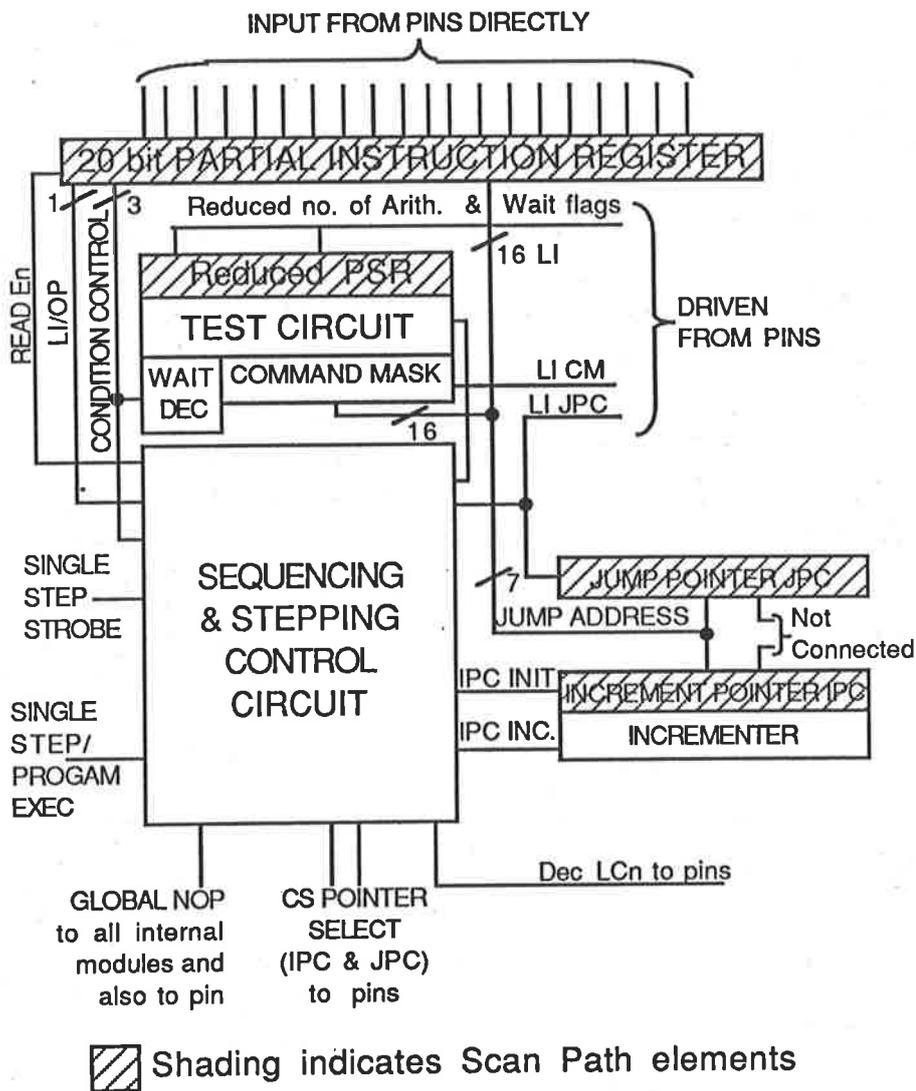


Figure B.4: Reduced Control Core MPC Configuration.

Appendix C

Scan Path Yield.

Because this chip is a prototype, rather than a production item, the causes of failure must be isolated where possible, and furthermore, wherever possible the chip must be capable of partial operation even in the presence of faulted modules. This impinges directly on the scan path design, because these are vulnerable to single point failures, and yet they are required in many areas of the chip to give controllability and observability to otherwise untestable modules.

The scan designs must try to attain the following goals:

- to maximise the absolute yield or reliability of the complete system of scan paths,
- to maximise the functionality of the scan system in the presence of faults,
- to limit the complexity of the total scan system to a level supportable by existing chip facilities, and

- where this does not counter the first two aims, to minimise the scan system area overheads.

Heuristically, if a single path suffers a single failure, it produces no data whatsoever, because its output cannot be verified. On the other hand, a multiple multiplexed path suffers a total failure only if one of the few critical components is disabled: an open circuit pin, or a multiplexor failure such that none of the lines is connected correctly. In the case of any other single failure, the branch of scan path faulted will yield no data, but all other branches are still capable of applying and capturing data.

With regards to multiple faulting, even under the assumption of random defect distribution the multiple branch scan path accrues advantages over the single path, while the phenomenon of defect clustering is likely to further improve this, since if a branch contain one fault, the clustering makes increases the likelihood of a second fault in that branch and decreases the likelihood of faults outside the clustering “neighbourhood”, and hence decreases the likelihood of another branch failing. This particular facet will not be considered in the following analysis, as the clustering mechanisms are not fully understood or quantified.

C.1 Analysis of Yield.

Consider a system in which N elements must be connected into a scan path, or multiple scan paths. A number of assumptions must be made before analysis can proceed.

1. Each scan path element is assumed to have equal probability of failure, and these probabilities are assumed to be independent.
2. The multiplexor elements, if required, are assumed to have probabilities of failure that are independent of the each other, and of the scan paths.
3. It is assumed that, on any given path, a single fault reduces the functionality of that path to zero, as the data cannot (in general) be verified.
4. Assume that the input pins contribute a negligible failure probability.

C.1.1 A Single Long Scan Path.

Consider first the case where all N elements are connected into a single path. If each element has a probability of failure p , where

$$0 < p \ll 1$$

then the probability of an intact scan path is given by

$$Pr(\textit{Intact Scan Path}) = (1 - p)^N$$

and the probability of a defective scan path by

$$Pr(\textit{Defective Scan Path}) = 1 - (1 - p)^N$$

Since any break is assumed to reduce functionality to zero,

$$Pr(\textit{Partial Functionality}) = Pr(\textit{Intact Scan Path}) = (1 - p)^N$$

Define

$$\begin{aligned} P_{PF/SP} &= Pr(\text{Partial Functionality/Single Scan Path}) \\ &= (1 - p)^N \end{aligned}$$

C.1.2 Multiplexed Multiple Scan Paths.

Consider the N elements to be connected into two scan paths, each of length $N/2$. The two scan paths share common input and output pins, and are multiplexed together at both ends by cells with equal and independent probabilities of failure q , where

$$0 < q \ll 1$$

Then the probability of all cells working correctly is

$$Pr(\text{No Failures}) = (1 - q)^2 \times (1 - p)^N$$

This is marginally less than the corresponding probability for the single scan path. However, consider the case when failures do occur. In the single path case, this results in no functionality. In the double path case, one path may be defective while the other is still functional. The only single faulty cell that may reduce the functionality to zero is the multiplexor cell.

$$\begin{aligned} P_{PF/DP} &= Pr(\text{Partial Functionality/Double Scan Path}) \\ &= Pr(2 \text{ good multiplexors and at least one good path}) \\ &= Pr(2 \text{ good multiplexors}) \times Pr(\text{at least one good path}) \\ &= Pr(2 \text{ good multiplexors}) \times (1 - Pr(\text{both paths broken})) \end{aligned}$$

$$\begin{aligned}
&= \text{Pr}(2 \text{ good multiplexors}) \times (1 - \text{Pr}(\text{one path broken})^2) \\
&= \text{Pr}(2 \text{ good multiplexors}) \times (1 - (1 - \text{Pr}(\text{one path good}))^2) \\
&= (1 - q)^2 (1 - (1 - (1 - p)^{N/2})^2) \\
&= (1 - q)^2 (1 - (1 + (1 - p)^N - 2(1 - p)^{N/2})) \\
&= (1 - q)^2 (2(1 - p)^{N/2} - (1 - p)^N)
\end{aligned}$$

By comparing the two probabilities for partial functionality, the conditions under which substitution of a double multiplexed path for a single path results in an increase in the probability of partial functionality may be deduced. An increase requires

$$\begin{aligned}
P_{PF/DP} - P_{PF/SP} &= (1 - q)^2 (2(1 - p)^{N/2} - (1 - p)^N) - (1 - p)^N \\
&= (1 - p)^{N/2} (2(1 - q)^2 - (1 - p)^{N/2} (1 - q)^2 - (1 - p)^{N/2}) \\
&= (1 - p)^{N/2} (1 - q)^2 (2 - (1 - p)^{N/2} - \frac{(1 - p)^{N/2}}{(1 - q)^2}) \\
&> 0
\end{aligned}$$

But

$$0 < (1 - p)^{N/2} < 1 \quad \forall N,$$

and

$$0 < (1 - q)^2 < 1$$

so the requirement for an improvement becomes

$$2 - (1 - p)^{N/2} - \frac{(1 - p)^{N/2}}{(1 - q)^2} > 0$$

This is satisfied if

$$1 - \frac{(1 - p)^{N/2}}{(1 - q)^2} > 0$$

or, rearranging,

$$(1 - p)^{N/2} < (1 - q)^2$$

Reinterpreting this in terms of the components, positive improvement occurs if

$$Pr(a \text{ good chain}) < Pr(2 \text{ good multiplexors})$$

To summarise the result, a gain in probability of partial functionality results from the use of double multiplexed paths, rather than a single path, if a failure is more likely in the half length chain than in the multiplexors. This can be generalised to apply other configurations of paths, and to splitting a path into two with separate output pins.

In the example that follows, using typical design values from TFB, the probabilities p and q are assumed equal, a fair estimate in that the dimensions and complexities of the two cells are roughly comparable.

C.2 An Sample Calculation.

Laid out scan path cell has a bounding box of about 17x17 virtual grid lines.

Compaction results in about 6λ per grid: cell area $\approx 10^4\lambda^2$.

Assume a process of 2 micron feature size: $\lambda = 1$ micron.

Cell Area $A \approx 10^{-4}cm^2$

Defect Densities (typical American processes) $D \approx 2-10$ defects/ cm^2 .

Yield (using Stapper's model [126]):

$$Yield Y = \frac{1}{1 + AD}$$

$$\begin{aligned}
&= \frac{1}{1 + 10^{-4} \times (2 - 10)} \\
&= 0.99980 - 0.99900
\end{aligned}$$

Assume the multiplexor and scan path cells have equal failure probability.

$$Pr(\text{failure of cell}) = 1 - Y = 0.00020 - 0.00100$$

Say 128 cells to examine: $N = 128$. Take $Y = 0.99950$ as an example.

$$\begin{aligned}
P_{PF/DP} - P_{PF/SP} &= (1 - p)^{N/2}(1 - q)^2(2 - (1 - p)^{N/2}) - \frac{(1 - p)^{N/2}}{(1 - q)^2} \\
&= (0.99950)^{128/2}(0.99950)^2(2 - (0.99950)^{128/2}) - \frac{(0.99950)^{128/2}}{(0.99950)^2} \\
&= (0.99950)^{66}(2 - (0.99950)^{64}) - (0.99950)^{62} \\
&= 0.06002
\end{aligned}$$

Improvement in probability of partial functionality = 6%. Given that

$$P_{PF/SP} = (1 - p)^N = (0.99950)^{128} = 0.93799$$

$$P_{PF/DP} = 0.93799 + 0.06002 = 0.99801$$

$$Pr(\text{No Faults/Double Multiplexed Path}) = (0.99950)^{128+2} = 0.93705$$

Summarising the above results, this example shows that, for the process feature sizes and defect densities expected to be used in TFB fabrication, the choice of multiplexed scan paths over a single path can result in significant improvements in the probability of partial scan path functionality (in this example a 6% improvement), with only a marginal down-grading of the overall scan system yield (in this case, the difference is 0.1%).

Appendix D

A Built In Test System for the Control Store.

At one stage during the development of the test procedures for this chip, a totally built-in tester was proposed for the Control Store. The idea was developed substantially before being discarded on the grounds that the extra complexity and area required for the implementation were not justified by the savings in test time. Nonetheless, the system is outlined here as an example of a possible method for built-in RAM testing.

The requirements for the RAM test applied by the built-in tester (BIT) are identical to those for the externally controlled test implemented. In addition, as well as detecting faulty rows, columns and cells, the BIT must also initiate the column and row sparing operations. The sparing was to be functionally equivalent to that now implemented, as described in Chapter 3, with four BARs and a row of column multiplexors controlled by a row of

master-slave latches. There is a difference in the method of initialising the pointers and BARs in the two cases: this will be explained later.

D.1 The Test Algorithm and Hardware.

The fault model for the memory cells is as described in Chapter 4: cells may be SA faulted, or API faulted in conjunction with a nearest column neighbour, but not with a row neighbour. Column faults may occur, and row faults, but these are not specifically examined for. Instead they are detected as a number of cell faults in one row or column. The reason for this is that the latter condition may arise purely from cell faults, and must be detected, so a systematic row or column fault will be tested by the checks for the cell faults.

The criterion for rejecting rows and columns, and replacing them with spares, needs examination. There are at most four available spare rows, so any column five or more faults (that are not otherwise spared out) must be replaced. Similarly, any row with two or more faults must be replaced, regardless of whether one of those faults may be removed by the use of the spare column.

During a built-in test, internal hardware must apply the test patterns to the CS array, receive the response and compare it with the expected value. This leads to the organisation of the test application and analysis circuits into bitslices situated under each column. The minimum circuitry required for this is a Write Register WR, which may be set or reset at the command of the BIT controller, an Instruction Register IR which captures

the column output, and some sort of comparison circuit in each column. As it is easier to test all columns at once before making the decision on which to discard, the WR, IR, comparison circuit and multiplexor row all comprise of 64 bitslices, rather than the 63 required functionally. The general arrangement of the system is illustrated in Figure D.1

It is possible to detect multiple row faults at test time by comparing the outputs of the column analysis circuits. This may be achieved by chaining results from column to column in a part of the column analysis circuits, and iteratively checking for multiple faults. Two flags, "Prior Fault", and "Prior Multiple Fault" may be generated by compounding results from column to column: the "Prior Fault" output from a column is simply the OR of the analysis output with the incoming "Prior Fault", while the "Prior Multiple Fault" output is the OR of the incoming "Prior Multiple Fault" with a "Local Multiple Fault" signal generated by ANDing the "Prior Fault" input with the analysis output. This is illustrated in Figure D.2. There may be some problems associated with the large gate delay across the array, but this could be overcome by idling for a period to let the chains settle.

If the BIT controller receives a "Multiple Fault" flag it may immediately organise the required row sparing operation. This is convenient, as the pointer which addressed the faulty row may be used to write the faulty location into a BAR. Also, if the multiply-faulted rows are removed from circuit immediately, the faults in these rows may be removed from consideration in the decision as to which column to spare out.

Detecting which columns are multiply faulted is not as trivial as detecting a multiply faulted column. A multiply faulted column is only detected

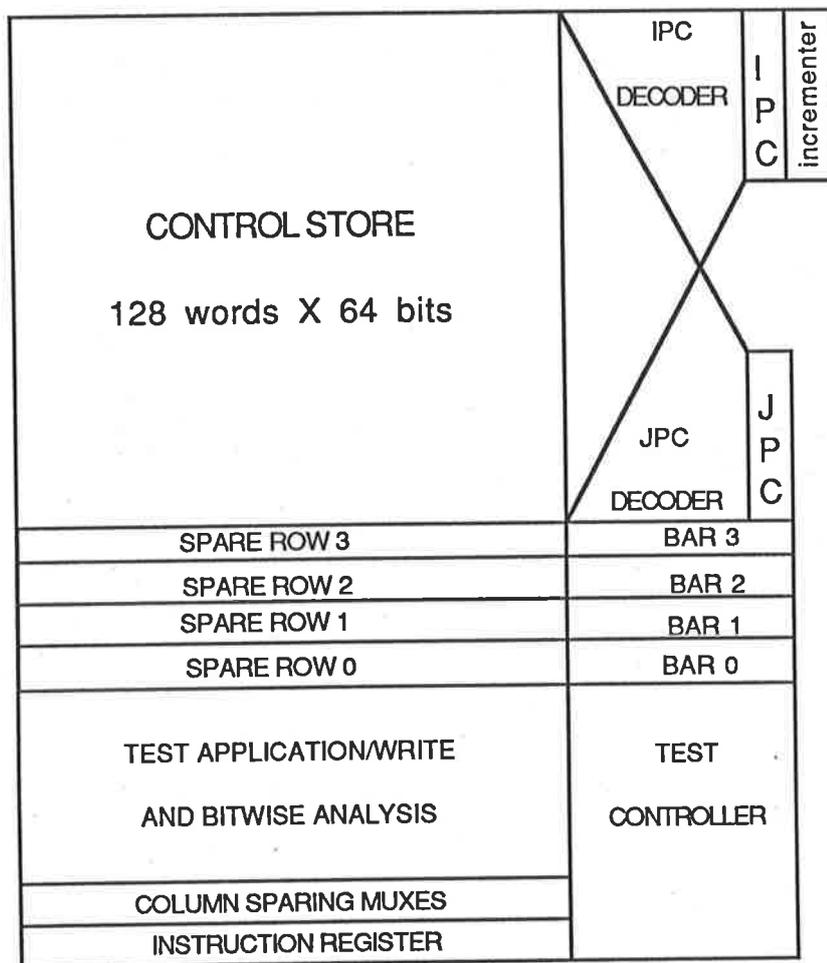
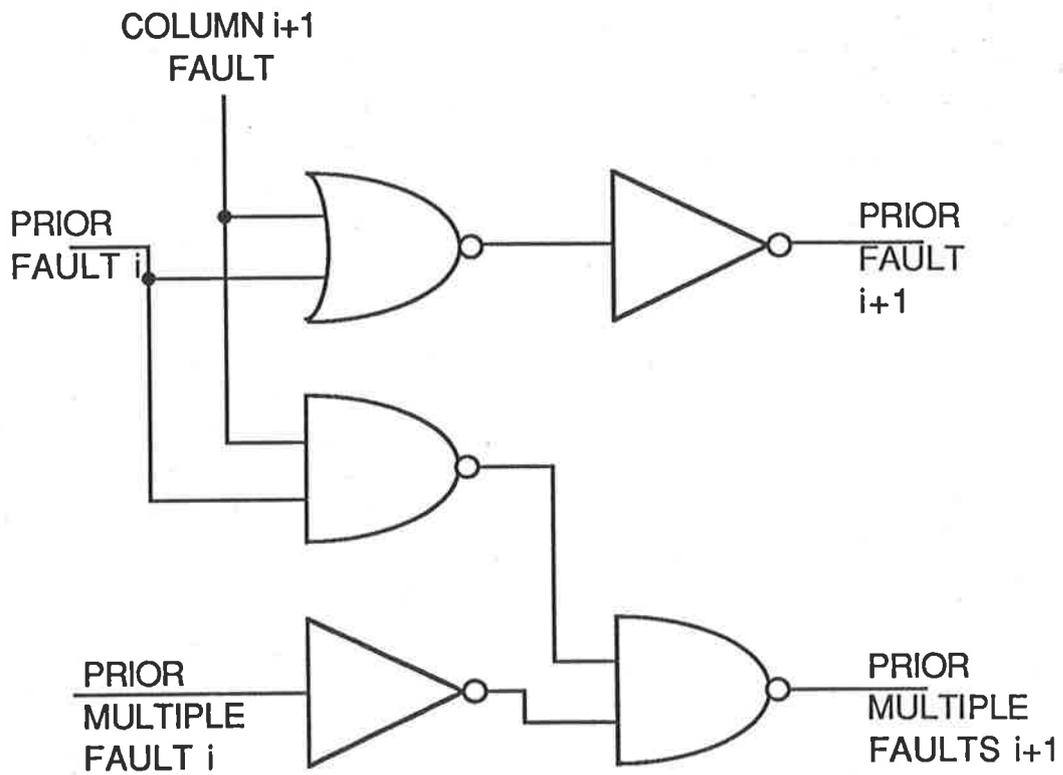


Figure D.1: General Arrangement for CS BIT Scheme.



$$PF_{i+1} = PF_i + F_{i+1}$$

$$MPF_{i+1} = MPF_i + PF_i \cdot F_{i+1}$$

Figure D.2: Iterative Flag Generation Circuit.

by applying all row tests to the array and counting the number of column faults which occur in that test. However, the count should not include faults which are part of a multiple row fault, as these must be removed by row sparing, and hence are not of interest. Furthermore, it is only necessary for a column to have five (unspared) faults before it must be discarded.

The detection may be done by counters or incrementers located in each column analysis block. Three-bit counters will suffice if they each have a feedback lock, disabling further incrementation once a count of five is reached. However, it is more economical to utilise a 5 bit shift register, simply shifting in a one each time a column fault is counted. This requires no locking feedback. Furthermore, this shift chain is useful later in the interrogation stage.

If the BIT controller broadcasts an "Increment" signal on receipt of a Single Fault flag (but not a Multiple Fault flag) from the column iterative analysis chain, the counters may AND this with the fault status of their particular column to determine whether to increment (*i.e.* shift) or not. After a complete pass through the cell test for the CS array, the multiple row faults should be removed from circuit, and each column counter will contain the number of cell faults for that column, up to a maximum of five, indicated by the number of bit positions filled.

At this stage the column counters may be interrogated to determine which has the highest count of faults. The idea is to set off an interrogating pulse to the first column, where the count shifter's MSB bit position is checked: if it is filled, then the multiplex latch is set, removing that column, but if not, the pulse is clocked to the next column to initiate the

column check. This is repeated until the pulse has had time to propagate right across the array, if it does not match the count at any stage. The controller interprets the pulse return as a mismatch, and organises a shift on all the count shifters, continuing to fill the LSBs with ones. This results in the counters all containing one more than their fault count. The interrogating pulse is set off again, attempting to match the condition (Fault Count + 1 = 5). This procedure is reiterated until a match occurs. This is guaranteed, even in the case of a fault-free array, because of the ones being fed in at the LSB of the shifters: in the fault-free array case, the first column interrogated will be switched out eventually.

The general arrangement of the test application and analysis circuitry in any one column is illustrated in Figure D.3. The system described so far is capable of organising the column switching, but the overall sequencing, and the actual tests applied to each row, needs detailing. The details of the initialisation of the row sparing BARs have never been fully clarified, but some form of parallel write from IPC is the most likely solution.

Before considering the actual form of the row test, note first the functional requirements for the CS pointers IPC and JPC. IPC may be loaded as a consequence of a JPC load, but normally increments by one. JPC is loadable under program control. This restricts the addressing sequences that may readily be performed.

The API fault model in use requires the examination of the cells above and below the cell under test to determine whether any data-dependent interaction occurs between them and the cell under test. Accessing the cell above the one most recently accessed is not difficult, using the IPC incre-

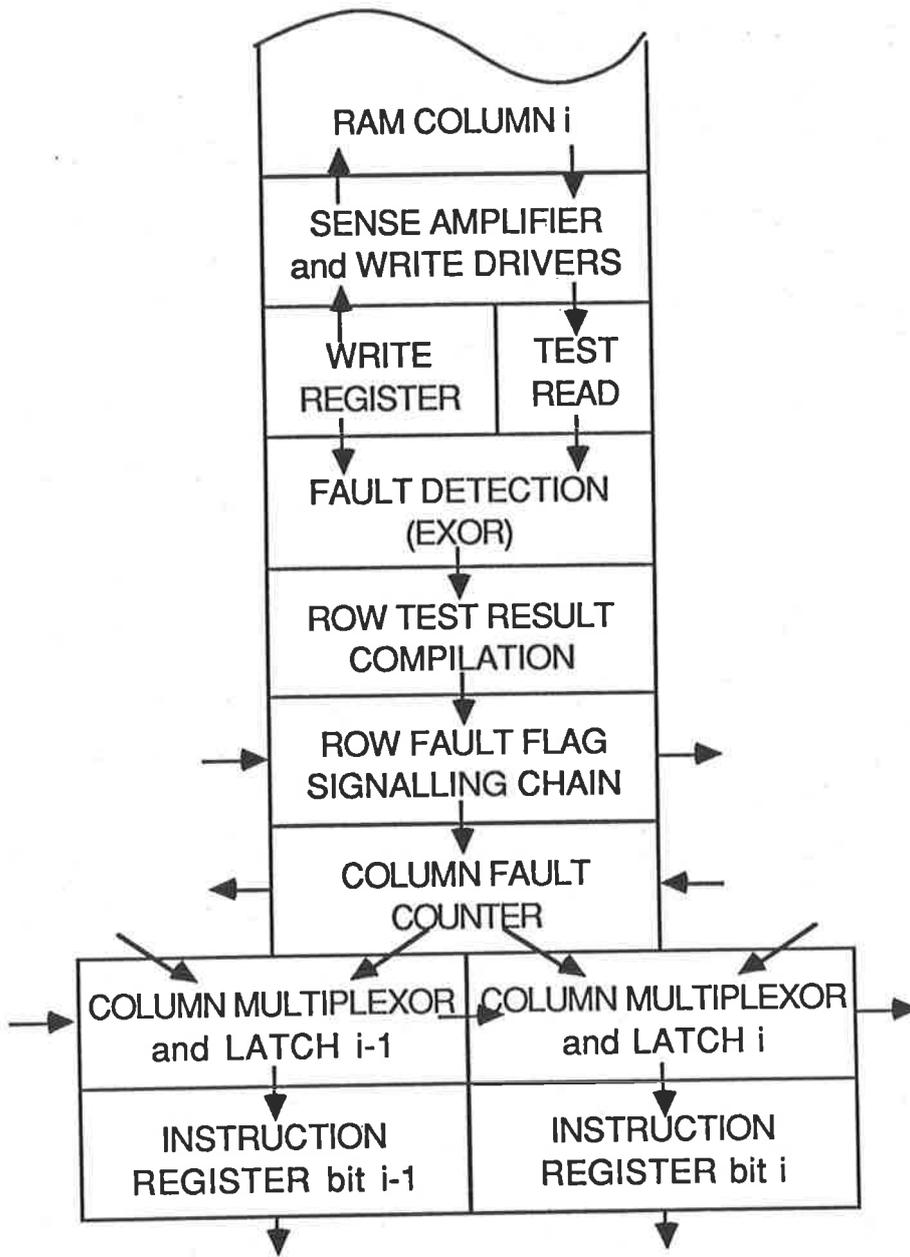


Figure D.3: Arrangement of Column Circuits.

ment facility, but to access the one below is not so easy. It probably requires the use of JPC to hold the previous value of IPC, a requirement completely divorced from, and in fact quite the opposite of the normal functionality. The details of the addressing mechanisms had not been worked out entirely when this approach was abandoned.

With regards to the test patterns, all that is required is the all zero and the all one pattern, because the columns are assumed to be independent and not subject to API faults. The order of applying tests and reading results directly affects the compactness of the test. The most compact test sequence devised while examining this problem is summarised in Table D.1. The sequence is specified for one pair of cells, with the complete sequence being generated by repeating the tabulated sequence for increasing values of *i*, the word address. The notation in the Fault column of this table is interpreted as follows: the cell under test is given by the column in which the result is written, the value preceded by the "@" symbol is the FAULTED value of the cell under test, and the other value, on the left or on the right within the bracket, is the value of the cell below or above, respectively, which specifies the condition for an API test.

If the tabulated test is used (bearing in mind that the pointer control has not been clarified entirely) one feature is of importance: for every Read, the expected value is that which was most recently written from the Write register, even if not to the same address. This allows the fault detection at any particular column to be carried out simply by taking the EXclusive OR of WR and IR, which will return a one if a fault is detected after a read.

| Step No. | R/W, Cell | | Value (after op.) | | Fault Tested | |
|---|-----------|--------------|-------------------|--------------|--------------|--------------|
| | Cell i | Cell $i + 1$ | Cell i | Cell $i + 1$ | Cell i | Cell $i + 1$ |
| 1 | | W | 0 | 0 | | |
| 2 | R | | 0 | 0 | (@1,0) | |
| 3 | W | | 1 | 0 | | |
| 4 | R | | 1 | 0 | (@0,0) | |
| 5 | | W | 1 | 1 | | |
| 6 | R | | 1 | 1 | (@0,1) | |
| 7 | W | | 0 | 1 | | |
| 8 | R | | 0 | 1 | (@1,0) | |
| This completes testing of cell i Next section tests cell $i + 1$ | | | | | | |
| 9 | | W | 0 | 0 | | |
| 10 | | R | 0 | 0 | | (0,@1) |
| 11 | | W | 0 | 1 | | |
| 12 | | R | 0 | 1 | | (0,@0) |
| 13 | W | | 1 | 1 | | |
| 14 | | R | 1 | 1 | | (1,@0) |
| 15 | | W | 1 | 0 | | |
| 16 | | R | 1 | 0 | | (1,@1) |

Table D.1: CS Test Sequence.

D.2 Reasons For Discarding This Design.

There are several reasons why this design was not adopted. First, the work done to date indicates that this solution is very expensive in terms of area overhead for testability. Rough estimates, which probably underestimate the BIT controller complexity, suggest an overhead of some eleven thousand transistors. As a percentage of the transistor count of the CS, this is 18 %, while area estimates extrapolated from typical layouts indicates that the test circuitry would have occupied some 6% of the total chip area.

Furthermore, estimates of the total test time for the CS using this method indicated that about 14000 clock cycles would be required, whereas an initial estimate of the scan path test time was about 70000 clock cycles: a large design effort was required, together with a significant increase in the overall chip complexity, to provide a factor of five decrease in the test time for one partition, but in absolute terms that decrease was from 8.8 milliseconds to 1.7 milliseconds . This time decrease was considered to be an insufficient return for the area and complexity penalty associated with the BIT scheme.

Another consideration was that the BIT scheme seemed to be heading for some rather complex problems of routing and connectivity in the area of the pointers and BARs, an area which is already crowded by the functional connections required.

It was also realised that individual cells may fail one test but not another, and that if two cells in a particular row failed there was no guarantee that they would be detected by the same test: *e.g.* one may be detected as

SA1 and the other as SA0. To recognise this as a multiple row fault is beyond the scope of the system described above, and alterations to the scheme to overcome this deficiency would result in further test time, complexity, and area overhead.

Yet another problem was the difficulty of testing the tester: some portions could be made scan-testable, but this seemed a little pointless in the light of the efforts in other areas to keep testing on chip.

The final point against this scheme is that it does not interface or mesh well with the tests required for the other portions of the control core. In fact it works against that aim by substantially increasing the complexity of significant portions of the control core, notably the pointer interconnections. On the other hand the test finally implemented, based on a wide network of scan paths, is a very well integrated test sequence, with in many places two or more different tests being carried out by the same instruction, in such a way that the results of each are distinguishable.

Bibliography

- [1] M. S. Abadir and H. K. Reghbati, "Test Generation for LSI: A Case Study", *Proc. 21st Design Automation Conference*, pp. 180-190, June 1984.
- [2] J. A. Abraham and D. D. Gajski, "Design of Testable Structures Defined by Simple Loops", *IEEE Trans. Comp.*, vol. C-30, no. 11, November 1981, pp. 875-883.
- [3] J. A. Abraham and S. M. Thatte, "Fault Coverage of Test Programs for a Microprocessor", *Digest 1979 IEEE Test Conference*, pp. 18-22, 1979.
- [4] V. K. Agarwal and A. S. Fung, "Multiple fault cover with single fault detecting test sets", *IEEE Trans. Comp.*, vol. C-30, no. 11, November 1981, pp. 855-865.
- [5] V. D. Agrawal and M. R. Mercer, "Testability Measures - What Do They Tell Us?", *Proc. 1982 International Test Conference*, Philadelphia, Pa., November 1982, pp. 391-396.
- [6] H. Ando, "Testing VLSI with Random Access Scan", *Digest IEEE Computer Society International Conference COMPCON 80*, pp. 50-52, February 1980.
- [7] I. Athanasiou and C. Zervos, "A Method for Deriving an Optimal Test Strategy for Microprocessors", *Proc. 13th Annual International Symposium on Fault Tolerant Computing*, pp. 378-381, 1983.
- [8] T. P. Barnwell III and C. J. M. Hodges, "Optimal Implementation of Signal Flow Graphs on Synchronous Multiprocessors", *Proc. IEEE International Conference on Parallel Processing*, pp. 90-95, August 1982.
- [9] W. Barraclough, A. C. L. Chiang and W. Sohl, "Techniques for Testing the Microcomputer Family", *Proc. IEEE*, vol. 64, no. 6, June 1976, pp. 943-950.
- [10] Z. Barzilai, J. Savir, G. Markowsky and M. G. Smith, "VLSI Self-Testing Based on Syndrome Techniques", *Digest 1981 IEEE Test Conference*, pp. 102-109, 1981.

- [11] Z. Barzilai, J. Savir, G. Markowsky and M. G. Smith, "Weighted Syndrome Sums Approach to VLSI Testing", *IBM Technical Disclosure Bulletin*, vol. 24 no. 9, February 1982, pp. 4494-4496.
- [12] C. Bellon et al., "Automatic Generation of Microprocessor Test Programs", *Proc. 19th Design Automation Conference*, pp. 566-573, June 1982.
- [13] C. Bellon and R. Velazco, "Taking Into Account Asynchronous Signals in Functional Test of Complex Circuits", *Proc. 21st Design Automation Conference*, pp. 490-496, June 1984.
- [14] R. G. Bennetts, C. M. Maunder and G. D. Robinson, "Camelot: A Computer Aided Measure for Logic Testability", *Proc. 1980 International Conference on Circuits and Computers ICCS 80*, pp. 1162-1165, October 1980.
- [15] N. Benowitz D. F. Calhoun, G. E. Alderson, J. E. Bauer and C. T. Joeckel, "An Advanced Fault Isolation System for Digital Logic", *IEEE Trans. Comp.*, vol. C-24 no. 5, May 1975, pp. 489-497.
- [16] D. K. Bhavsar and R. W. Heckelman, "Self-Testing By Polynomial Division", *Digest 1981 IEEE Test Conference*, pp. 208-216, 1981.
- [17] P. S. Bottorff, R. E. France, N. H. Garges and E. J. Orosz, "Test Generation for Large Logic Networks", *Proc. 14th Design Automation Conference*, pp. 479-485, June 1977.
- [18] D. Brahme and J. A. Abraham, "Functional Testing of Microprocessors", *IEEE Trans. Comp.*, vol. C-33 no. 6, June 1984, pp. 475-485.
- [19] F. Brglez, "Testability in VLSI", *Proc. 1983 Canadian Conference on VLSI*, University of Waterloo, pp. 90-95, October 1983.
- [20] F. Brglez, "On Testability Analysis of Combinational Networks", *Proc. 1984 IEEE International Symposium on Circuits and Systems*, pp. 221-225, May 1984.
- [21] F. Brglez, "Applications of Testability Analysis: From ATPG to Critical Delay Path Tracing", *Proc. 1984 International Test Conference*, Philadelphia, Pa., October 1984.

- [22] M. A. Breuer and A. D. Freidman, *Diagnosis and Reliable Design of Digital Systems*, Computer Science Press, 1976.
- [23] W. C. Carter, "The Ubiquitous Parity Bit", *Proc. 12th Annual International Symposium on Fault Tolerant Computing*, pp. 289-296, 1982.
- [24] W. C. Carter, "Signature Testing with Guaranteed Bounds for Fault Coverage", *Proc. 1982 International Test Conference*, Philadelphia, Pa., November 1982, pp. 75-82.
- [25] R. Chandramouli, "On Testing Stuck-Open Faults", *Proc. 13th Annual International Symposium on Fault Tolerant Computing*, pp. 258-265, 1983.
- [26] J. Chavade, M. Vergniault, P. Rousseau, Y. Crouzet and C. Landrault, "A Monolithic Self-Checking Error Detection Processor", *Digest 1980 IEEE Test Conference*, pp. 279-286, 1980.
- [27] C. H. Chen, "VLSI Design for Testability", *Digest 1979 IEEE Test Conference*, pp. 306-309, 1979.
- [28] C. H. Chen, "Designing Testable Synchronous Logic", *Digest 1981 IEEE Test Conference*, pp. 89-94, 1981.
- [29] C. H. Chen, "Design Methodology for Testable VLSI", *Digest 23rd IEEE Computer Society International Conference COMPCON Fall 81*, pp. 168-173, 1981.
- [30] K. W. Chiang and Z. G. Vranesic, "On Fault Detection in CMOS Logic Networks", *Proc. 20th Design Automation Conference*, pp. 50-56, 1983.
- [31] R. J. Clarke, P. Arya, R. J. Potter, G. J. Smith and I. D. Smith, "Hierarchical Verification and Incremental Test of a 100,000 Transistor Integrated Circuit", *Custom Integrated Circuits Conference*, Rochester, New York, 21-23 May 1984.
- [32] B. Courtois, "A Methodology for On-line Testing of Microprocessors", *Proc. 11th Annual International Symposium on Fault Tolerant Computing*, pp. 272-274, 1981.

- [33] W. Daehn and J. Mucha, "A Hardware Approach to Self-Testing of Large Programmable Logic Arrays", *IEEE Trans. Comp.*, vol. C-30, no. 11, November 1981, pp. 829-833.
- [34] R. P. Davidson, M. L. Harrison and R. L. Wadsack, "BELLMAC-32: A Testable 32 Bit Microprocessor", *Digest 1981 IEEE Test Conference*, pp. 15-20, 1981.
- [35] A. G. Dickinson and J. E. Rockliff, "The TFB Working Specification", Technical Report, Department of Electrical and Electronic Engineering, University of Adelaide. Version 1, December 1983.
- [36] E. B. Eichelberger, E. J. Muehldorf, R. G. Walter and T. W. Williams, "A Logic Design Structure for Testing Internal Arrays", *Proc. 3rd USA-Japan Computer Conference*, San Francisco, October 1978, pp. 266-272.
- [37] E. B. Eichelberger and T. W. Williams, "A Logic Design Structure for LSI Testability", *Proc. 14th Design Automation Conference*, pp. 462-468, June 1977.
- [38] C. Efstathiou and C. Halatsis, "Modular Realization of Totally Self-Checking Checkers for M-out-of-N Codes", *Proc. 13th Annual International Symposium on Fault Tolerant Computing*, pp. 145-157, 1983.
- [39] Y. M. Elziq, 1981, "Automatic Test Generation for Stuck-Open Faults in CMOS VLSI", *Proc. 18th Design Automation Conference*, pp. 347-354, 1981.
- [40] K. Eshraghian, A. Dickinson, J. E. Rockliff, G. Zyner, R. E. Bogner, R. C. Bryant, W. G. Cowley, B. R. Davis, D. S. Fensom, "A New CMOS VLSI Architecture for Signal Processing", *Digest VLSI PARC*, Melbourne, May 1984.
- [41] K. Eshraghian, R. C. Bryant, A. Dickinson, D. S. Fensom, P. D. Franzon, M. T. Pope, J. E. Rockliff, G. Zyner, "The Transform and Filter Brick: A New Architecture for Signal Processing", *Proc. VLSI 85*, Japan, October 1985.
- [42] P. P. Fasang, "BIDCO, Built-In Digital Circuit Observer", *Digest 1980 IEEE Test Conference*, pp. 261-266, 1980.

- [43] P. P. Fasang, "Circuit Module Implements Practical Self-Testing", *Electronics*, May 19 1982, pp. 164-167.
- [44] Fairchild Test Group Systems Group, "Serial Test Module Specification", July 1982.
- [45] A. D. Friedman, "Diagnosis of Short-Circuit Faults in Combinational Circuits", *IEEE Trans. Comp.*, vol. C-23 no. 7, July 1974, pp. 746-752.
- [46] P. D. Franzon, "Investigations of Yield Analysis", Technical Report, Department of Electrical and Electronic Engineering, University of Adelaide.
- [47] R. A. Frohwerk, "Signature Analysis: A New Digital Field Service Method", *Hewlett-Packard Journal*, pp. 2-8, May 1977.
- [48] E. Fujiwara, "A Self-Testing Group Parity Prediction Checker and its Use for Built-In Testing", *Proc. 13th Annual International Symposium on Fault Tolerant Computing*, pp. 146-153, 1983.
- [49] H. Fujiwara, K. Kinoshita and H. Ozaki, "Universal Test Sets for Programmable Logic Arrays", *Proc. 10th Annual International Symposium on Fault Tolerant Computing*, pp. 137-142, 1980.
- [50] H. Fujiwara and K. Kinoshita, "Testing Logic Circuits with Compressed Data", *Proc. 8th Annual International Symposium on Fault Tolerant Computing*, pp. 108-113, 1978.
- [51] H. Fujiwara and K. Kinoshita, "A Design of Programmable Logic Arrays with Universal Test Sets", *IEEE Trans. Comp.*, vol. C-30 no. 11, November 1981, pp. 823-828.
- [52] H. Fujiwara and T. Shimono, "On the Acceleration of Test Generation Algorithms", *Proc. 13th Annual International Symposium on Fault Tolerant Computing*, pp. 98-105, 1983.
- [53] H. Fujiwara, "A New PLA Design for Universal Testability", *IEEE Trans. Comp.*, vol. C-33 no. 8, August 1984, pp. 745-750.
- [54] S. Funatsu, N. Wakatsuki and T. Arima, "Test Generation Systems in Japan", *Proc. 12th Design Automation Symposium*, pp. 114-122, 1975.

- [55] J. Galiay, Y. Crouzet and M. Vergniault, "Physical versus Logical Fault Models for MOS LSI Circuits: Impact on Their Testability", *IEEE Trans. Comp.*, vol. C-29 no. 6, June 1980, pp. 527-531.
- [56] H. Gepraegs and H. Tandjung, "Test Method for VLSI Semiconductor Chips", *IBM Technical Disclosure Bulletin*, vol. 23 no. 8, January 1981, pp. 3743-3744.
- [57] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Networks", *Proc. 10th Annual International Symposium on Fault Tolerant Computing*, pp. 145-151, October 1980.
- [58] P. Goel and B. C. Rosales, "PODEM-X: An Automatic Test Generation System for VLSI Logic Structures", *Proc. 18th Design Automation Conference*, pp. 260-268, 1981.
- [59] L. H. Goldstein and E. L. Thigpen, "SCOAP: Sandia Controllability/Observability Analysis Program", *Proc. 17th Design Automation Conference*, pp. 190-196, June 1980.
- [60] K. Gutfreund, "Integrating the Approaches to Structured Design for Testability", *VLSI Design*, October 1983, pp. 34-42.
- [61] S. Z. Hassan, D. J. Lu and E. J. McCluskey, "Parallel Signature Analyzers - Detection Capabilities and Extensions", *Digest IEEE Computer Society International Conference COMPCON Spring 83*, pp. 440-445, February 1983.
- [62] J. P. Hayes, "Transition Count Testing of Combinational Logic Circuits", *IEEE Trans. Comp.*, vol. C-27 no. 6, pp. 613-620, June 1976.
- [63] J. P. Hayes, *Computer Architecture and Organisation*, McGraw-Hill, 1978.
- [64] J. P. Hayes, "Testing Memories for Single-Cell Pattern-Sensitive Faults", *IEEE Trans. Comp.*, vol. C-29 no. 3, pp. 249-254, March 1980.
- [65] J. T. Healy, "Automatic Testing and Evaluation of Digital Integrated Circuits", Reston Publishing Company Inc., Reston, Virginia.

- [66] S. J. Hong and D. L. Ostapko, "FIT-PLA: A Programmable Logic Array for Functional Independent Testing", *Proc. 10th Annual International Symposium on Fault Tolerant Computing*, pp. 131-136, 1980.
- [67] K. A. Hua, J.-Y. Jou and J. A. Abraham, "Built-In Tests for VLSI Finite State Machines", *Proc. 14th Annual International Symposium on Fault Tolerant Computing*, pp. 292-297, 1984.
- [68] J. L. A. Hughes, E. J. McCluskey and D. J. Lu, "Design of Totally Self-Checking Comparators With An Arbitrary Number of Inputs", *Proc. 13th Annual International Symposium on Fault Tolerant Computing*, pp. 169-172, 1983.
- [69] R. J. Illman, "Self-Tested Data Flow Logic: A New Approach", *IEEE Design and Test of Computers*, vol. 2 no. 2, April 1985, pp. 50-58.
- [70] S. K. Jain and V. D. Agrawal, "STAFAN: An Alternative to Fault Simulation", *Proc. 21st Design Automation Conference*, pp. 18-23, June 1984.
- [71] M. G. Karpovsky and R. G. Van Meter, "An Approach to the Testing of Microprocessors", *Proc. 21st Design Automation Conference*, pp. 196-202, June 1984.
- [72] J. Knaizuk, Jr. and C. R. P. Hartmann, "An Optimal Algorithm for Testing Stuck-At Faults in Random Access Memories", *IEEE Trans. Comp.*, vol. C-26 no. 11, November 1977, pp. 1141-1144.
- [73] J. Khakbaz and E. J. McCluskey, "Concurrent Error Detection and Testing for Large PLA's", *IEEE Trans. Electron Devices*, vol. ED-29 no. 4, April 1982, pp. 756-764.
- [74] B. Könemann, J. Mucha and G. Zwiehoff, "Built-In Logic Block Observation Techniques", *Digest 1979 IEEE Test Conference*, pp. 37-41, 1979.
- [75] B. Könemann, J. Mucha and G. Zwiehoff, "Built-In Test for Complex Digital Integrated Circuits", *IEEE Journal of Solid-State Circuits*, vol. SC-15 no. 3, June 1980, pp. 315-318.

- [76] D. Komonytsky, "LSI Self-Test Using Level-Sensitive Scan Design and Signature Analysis", *Proc. 1982 International Test Conference*, Philadelphia, Pa., November 1982, pp. 414-424.
- [77] M. W. Levi, "CMOS Is Most Testable", *Digest 1981 IEEE Test Conference*, pp. 217-220, 1981.
- [78] E. Lindbloom, S. Matheson and J. Catanzaro, "VLSI Testing, Diagnostics, and Qualification", *Proc. 1982 IEEE International Conference on Circuits and Computers ICC 82*, pp. 341-345, September 1982.
- [79] G. V. Lukianoff, J. S. Wolcott and J. M. Morissey, "Electron-Beam Testing of VLSI Dynamic RAMS", *Digest 1981 IEEE Test Conference*, pp. 68-76, 1981.
- [80] T. E. Mangir and A. Avizienis, "Failure Modes for VLSI and Their Effect on Chip Design", *Proc. 1980 International Conference on Circuits and Computers ICC 80*, pp. 685-688, October 1980.
- [81] T. E. Mangir, "Sources of Failure and Yield Improvement for VLSI and Restructurable Interconnects for RVLSI and WSI: Part I - Sources of Failures and Yield Improvement for VLSI", *Proceedings of the IEEE*, vol. 72 no. 6, June 1984.
- [82] P. C. Maxwell, "Design for Testability", VLSI Program, Division of Computing Research, Commonwealth Scientific and Industrial Research Organisation, Adelaide, South Australia, Technical Report VLSI-TR-83-1-2.
- [83] P. C. Maxwell, "Built-In Self Test: An Aid for Testing Complex Systems", *Proc. IREE/IEAust Microelectronics Conference: "Migrating Systems to Silicon"*, Sydney, 1985, pp. 128-135.
- [84] E. J. McCluskey, "Verification Testing - A Pseudoexhaustive Test Technique", *IEEE Trans. Comp.*, vol. C-33 no. 6, June 1984, pp. 541-546.
- [85] E. J. McCluskey, "Built-In Test Structures", *IEEE Design and Test of Computers*, vol. 2 no. 2, April 1985, pp. 21-28.
- [86] E. J. McCluskey, "Built-In Test Structures", *IEEE Design and Test of Computers*, vol. 2 no. 2, April 1985, pp. 29-36.

- [87] C. A. Mead and L. A. Conway, *Introduction to VLSI Systems*, Addison-Wesley, 1980.
- [88] K. C. Y. Mei, "Bridging and Stuck-At Faults", *IEEE Trans. Comp.*, vol. C-23 no. 7, July 1974, pp. 720-727.
- [89] Y. Min and S. Y. H. Su, "Testing Functional Faults in VLSI", *Proc. 19th Design Automation Conference*, pp. 384-392, 1982.
- [90] J. J. Moser, "LSSD Test Architecture", *IBM Technical Disclosure Bulletin*, vol. 24 no. 3, August 1981, pp. 1666-1667.
- [91] J. P. Mucha, "Hardware Techniques for Testing VLSI Circuits Based on Built-In Test", *Digest IEEE Computer Society International Conference COMPCON Spring 81*, pp. 366-369, 1981.
- [92] J. P. Mucha, "VLSI Testing - Problems and Solutions", *VLSI 85: Proc. International Conference on Very Large Scale Integration*, Tokyo, Japan, August 1985, edited by E. Horbst, pp. 359-366.
- [93] J. P. Mucha and W. Daehn, "Hardware Test Pattern Generation for Built-In Testing", *Digest 1981 IEEE Test Conference*, pp. 110-113, 1981.
- [94] J. C. Mudge and R. J. Clarke, "Australia's First Multi-Project Chip Implementation System", *Preprints, National Conference on Microelectronics 1982*, Adelaide, 12-14 May 1982, pp. 72-77.
- [95] J. C. Mudge, R. J. Clarke, M. L. Paltridge and R. J. Potter, "The Results of AUSMPC 5/82", *VLSI Design*, January/February 1983, pp. 52-56.
- [96] N. W. Murray, "Data Memory Control Structure for the TFB", Final Year Undergraduate Project Report, Department of Electrical and Electronic Engineering, University of Adelaide. November 1984.
- [97] J. C. Muzio and D. M. Miller, "Spectral Techniques for Fault Detection", *Proc. 12th Annual International Symposium on Fault Tolerant Computing*, pp. 297-302, 1982.
- [98] H. J. Nadig, "Signature Analysis - Concepts, Examples, and Guidelines", *Hewlett-Packard Journal*, pp. 15-21, May 1977.

- [99] R. Nair, S. M. Thatte and J. A. Abraham, "Efficient Algorithms for Testing Semiconductor Random-Access Memories", *IEEE Trans. Comp.*, vol. C-27 no. 6, June 1978, pp. 572-576.
- [100] T. Nanya and Y. Tohma, "A 3-Level Realization of Totally Self-Checking Checkers for M-out-of-N Codes", *Proc. 13th Annual International Symposium on Fault Tolerant Computing*, pp. 173-176, 1983.
- [101] S. Bozorgui-Nesbat and E. J. McCluskey, "Design for Autonomous Test", *Digest 1980 IEEE Test Conference*, pp. 15-21, 1980.
- [102] S. Bozorgui-Nesbat and E. J. McCluskey, "Design for Autonomous Test", *IEEE Trans. Comp.*, vol. C-30 no. 11, November 1981, pp. 866-874.
- [103] M. C. Obst, "CMOS RAM Structure Generator", Final Year Undergraduate Project Report, Department of Electrical and Electronic Engineering, University of Adelaide. November 1984.
- [104] R. Parthasarathy and S. M. Reddy, "A Testable Design of Iterative Logic Arrays", *IEEE Trans. Comp.*, vol. C-30 no. 11, November 1981, pp. 833-841.
- [105] C. C. Perkins, S. Sangani, H. Stopper and W. Valitski, "Design for In-Situ Chip Testing with a Compact Tester", *Digest 1980 IEEE Test Conference*, pp. 29-41, 1980.
- [106] S. Piestrak, "Design Method of totally Self-Checking Checkers for M-Out-Of-N Codes", *Proc. 13th Annual International Symposium on Fault Tolerant Computing*, pp. 162-165, 1983.
- [107] D. R. Resnick, "Testability and Maintainability with a New 6K Gate Array", *VLSI Design*, March/April 1983, pp. 34-38.
- [108] J. B. Rosenberg, "Chip Assembly Techniques for Custom IC Design in a Symbolic Virtual Grid Environment", *Conference on Advanced Research in VLSI*, MIT. 1984.
- [109] J. P. Roth, "Diagnosis of Automata Failures: A Calculus and A Method", *IBM Journal of Research and Development*, no. 10, pp. 278-281, October 1966.

- [110] L. P. Rubinfeld, "A Proof of the Modified Booth's Algorithm for Multiplication", *IEEE Trans. Comp.*, October 1975, pp. 1014-1015.
- [111] K. K. Saluja, K. Kinoshita and H. Fujiwara, "An Easily Testable Design of Programmable Logic Arrays for Multiple Faults", *IEEE Trans. Comp.*, vol. C-32 no. 11, November 1983, pp. 1038-1046.
- [112] K. K. Saluja and J. S. Upadhyaya, "Divide and Conquer Strategy for Testable Design of Programmable Logic Arrays", *Proc. IREE/IEAust Microelectronics Conference: "Migrating Systems to Silicon"*, Sydney, 1985, pp. 121-127.
- [113] E. Savio, "Input and Output Modules for TFB", Final Year Undergraduate Project Report, Department of Electrical and Electronic Engineering, University of Adelaide. November 1984.
- [114] J. Savir, "Syndrome-Testable Design of Combinational Circuits", *IEEE Trans. Comp.*, vol. C-29 no. 6, June 1980, pp. 442-451.
- [115] J. Savir, "Syndrome-Testing of 'Syndrome-Untestable' Combinational Circuits", *IEEE Trans. Comp.*, vol. C-30 no. 8, August 1981, pp. 606-608.
- [116] J. Savir, P. H. Bardell and G. Ditlow, "Random Pattern Testability", *Proc. 13th Annual International Symposium on Fault Tolerant Computing*, pp. 80-89, 1983.
- [117] J. Savir, P. H. Bardell and G. Ditlow, "Random Pattern Testability", *IEEE Trans. Comp.*, vol. C-33 no. 1, January 1984, pp. 79-90.
- [118] M. T. M. Segers, "A Self-Test Method for Digital Circuits", *Digest 1981 IEEE Test Conference*, pp. 79-85, 1981.
- [119] Report 1 of the Signal Processing Sub-Group, CMOS VLSI Project, Department of Electrical and Electronic Engineering, University of Adelaide, June 1983.
- [120] J. E. Smith, "Measure of the Effectiveness of Fault Signature Analysis", *IEEE Trans. Comp.*, vol. C-29 no. 6, June 1980, pp. 510-514.
- [121] K. Son and D. K. Pradhan, "Completely Self-Checking Checkers in PLAs", *Digest 1981 IEEE Test Conference*, pp. 231-237, 1981.

- [122] V. P. Srimi, "API Tests for RAM Chips", *Computer*, July 1977, pp. 32-35.
- [123] V. P. Srimi, "Fault Location in a Semiconductor Random-Access Memory Unit", *IEEE Trans. Comp.*, vol. C-27 no. 4, April 1978, pp. 349-358.
- [124] T. Sridhar and J. P. Hayes, "Design of Easily Testable Bit-Sliced Systems", *IEEE Trans. Comp.*, vol. C-30 no. 11, November 1981, pp. 842-854.
- [125] T. Sridhar et al., "Analysis and Simulation of Parallel Signature Analyzers", *Proc. 1982 International Test Conference*, Philadelphia, Pa., November 1982, pp. 656-661.
- [126] C. H. Stapper et al., "Yield Model for Productivity Optimization of VLSI Memory Chips with Redundancy and Partially Good Product", *IBM Journal of Research and Development*, Vol. 24 no. 3, May 1980, pp. 389-412.
- [127] J. H. Stewart, "Future Testing of Large LSI Circuit Cards", *Digest 1977 IEEE Test Symposium*, pp. 6-17, 1977.
- [128] N. R. Strader II and T. J. Brosnan, "Error Detection for Serial Processing Elements in Highly Parallel VLSI Processing Architectures", *Proc. 1984 Conference on Advanced Research in VLSI*, M.I.T., January 1984, pp. 184-193.
- [129] S. Y. H. Su and T. Lin, "Functional Testing Techniques for Digital LSI/VLSI Systems", *Proc. 21st Design Automation Conference*, pp. 517-528, June 1984.
- [130] Z. Sun and L.-T. Wang, "Self-Testing of Embedded RAMS", *Proc. 1984 International Test Conference*, Philadelphia, Pa., October 1984, pp. 148-156.
- [131] A. K. Susskind, "Testing by Verifying Walsh Coefficients", *Proc. 11th Annual International Symposium on Fault Tolerant Computing*, pp. 206-208, 1981.
- [132] R. Treuer, H. Fujiwara and V. K. Agarwal, "Implementing a Built-In Self-Test PLA Design", "Built-In Test Structures", *IEEE Design and Test of Computers*, vol. 2 no. 2, April 1985, pp. 37-48.

- [133] F. F. Tsui, "In Situ Testability Design (ISTD) - A New Approach for Testing High-Speed LSI/VLSI Logic", *Proc. IEEE*, vol. 70, no. 1, January 1982, pp. 59-78.
- [134] R. L. Wadsack, "Fault Modelling and Logic Simulation of CMOS and MOS Integrated Circuits", *Bell System Technical Journal*, May-June 1978, pp. 1449-1474.
- [135] D. Wescott, "The Self-Assist Test Approach to Embedded Arrays", *Digest 1981 IEEE Test Conference*, pp. 203-207, 1981.
- [136] N. H. Weste and K. Eshraghian, *Principles of CMOS VLSI Design: A Systems Perspective*, Addison-Wesley, 1985.
- [137] M. J. Y. Williams and J. B. Angell, "Enhancing Testability of Large Scale Integrated Circuits via Test Points and Additional Logic", *IEEE Trans. Comp.*, vol. C-22 no. 1, January 1973, pp. 46-60.
- [138] T. Williams, "The History and Theory of the Stuck-at Fault", *Proc. 6th European Conference on Circuit Theory and Design*, 6-8 September 1983, pp. 80-81.
- [139] T. W. Williams and K. P. Parker, "Testing Logic Networks and Designing for Testability", *Computer*, vol. 12, no. 10, October 1979, pp. 9-21.
- [140] T. W. Williams and K. P. Parker, 1983, "Design for Testability - A Survey", *Proc. IEEE*, vol. 71, no. 1, January 1983, pp. 98-112.
- [141] P. Wohlfarth and D. Smith, "Testing the New Generation of Microprocessors", *Digest 1979 IEEE Test Conference*, pp. 255-259, 1979.
- [142] A. Yamada, S. Funatsu and M. Kawai, "Application of Shift Register Approach and Its Effective Implementation", *Digest 1980 IEEE Test Conference*, pp. 22-25, 1980.
- [143] A. Yamada, N. Wakatsuki, H. Shibano, O. Itoh, K. Tomita and S. Funatsu, "Automatic Test Generation for Large Digital Circuits", *Proc. 14th Design Automation Conference*, pp. 78-83, June 1977.
- [144] A. Yamada, N. Wakatsuki and S. Funatsu, "Designing Digital Circuits with Easily Testable Consideration", *Proc. 1978 IEEE Test Conference*, pp. 98-102, November 1978.

- [145] Y. You and J. P. Hayes, "A Self-Testing Dynamic RAM Chip", *Proc. 1984 Conference on Advanced Research in VLSI*, M.I.T., January 1984, pp. 159-168.
- [146] J. J. Zasio, "Shifting Away From Probes for Wafer Test", *Digest IEEE Computer Society International Conference COMPCON Spring 83*, pp. 395-398, February 1983.
- [147] G. B. Zyner, "An Arithmetic Logic Unit for the TFB", Technical Report, Department of Electrical and Electronic Engineering, University of Adelaide. Version 3, November 1984.
- [148] G. B. Zyner, "Revisions to the TFB I/O Processors", Technical Report, Department of Electrical and Electronic Engineering, University of Adelaide. Draft, February 1985.
- [149] G. B. Zyner, "Revisions to the TFB Data Memory Pointer Control Circuitry", Technical Report, Department of Electrical and Electronic Engineering, University of Adelaide. Draft, April 1985.
- [150] G. B. Zyner and K. Eshraghian, "Multiplier/Divider Designs for VLSI Signal Processing System", *Digest 20th International Electronics Convention of the Institution of Radio and Electronic Engineers Australia IRECON '85*, Melbourne, 30 Sept.-4 October 1985, pp. 253-256.