1

CANADIAN THESES ON MICROFICHE

I.S.B.N.

THESES CANADIENNES SUR MICROFICHE

National Library of Canada
Collections Development Branch

Canadian Theses on
Microfiche Service

Ottawa, Canada
K1A 0N4

Bibliothèque nationale du Canada
Direction du développement des collections

Service des thèses canadiennes
sur microfiche

NOTICE

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

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

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

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

**THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED**

AVIS

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

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

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

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

**LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE**

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Division

Division des thèses canadiennes

Ottawa, Canada
K1A 0N4

59930

PERMISSION TO MICROFILM — AUTORISATION DE MICROFILMER

- Please print or type — Écrire en lettres moulées ou dactylographier

Full Name of Author — Nom complet de l'auteur

STEVEN PETER MENZEL

Date of Birth — Date de naissance

NOVEMBER 29, 1954

Country of Birth — Lieu de naissance

CANADA

Permanent Address — Résidence fixe

87 GILMOUR ST., APT. 2A
OTTAWA, ONT.
K2P 0N5

Title of Thesis — Titre de la thèse

TESTABILITY ANALYSIS CONSIDERATIONS
OF DIGITAL CIRCUITS

University — Université

CARLETON UNIVERSITY

Degree for which thesis was presented — Grade pour lequel cette thèse fut présentée

MASTER OF ENGINEERING

Year this degree conferred — Année d'obtention de ce grade

1982

Name of Supervisor — Nom du directeur de thèse

JOHN KNIGHT

Permission is hereby granted to the NATIONAL LIBRARY OF CANADA to microfilm this thesis and to lend or sell copies of the film.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

L'autorisation est, par la présente, accordée à la BIBLIOTHÈQUE NATIONALE DU CANADA de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur se réserve les autres droits de publication, ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans l'autorisation écrite de l'auteur.

Date

Sept 14, 1982

Signature

S. Menzel

TESTABILITY ANALYSIS CONSIDERATIONS OF DIGITAL CIRCUITS

by

© Steven Peter Menzel

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

Master of Engineering

Department of Electronics
Faculty of Engineering
Carleton University
Ottawa, Canada

August, 1982

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

"Testability analysis considerations of digital circuits."

submitted by Steven Peter Menzel in partial fulfillment of
the requirements for the degree of Master of Engineering.

.....
Prof. J.P. Knight
Thesis Supervisor

.....
Prof. A.R. Boothroyd,
Chairman,
Department of Electronics

August, 1982

ABSTRACT

This thesis is concerned with the problems of estimating the degree of difficulty to be expected in testing a digital integrated circuit before the test program is actually generated, and attempting to analyze and locate potential problem areas before the circuit design is finalized. Some design for testability methods and testability analysis programs are considered. The SCOAP testability analysis algorithm and its limitations are discussed in detail. Several changes and improvements are proposed which will extend its capabilities.

ACKNOWLEDGEMENTS

I would personally like to express my gratitude to Dr. D.M. Caughey for his patience and insight, and to Dr. John Knight, my supervisor, for his guidance in this project. I also wish to thank Mr. Ion Ratiu of the University of California (Berkeley), and Dr. Franz Brglez of Northern Telecom Ltd., for their co-operation. I am also very grateful to the people at Mitel Corporation for the use of their computing and text editing facilities, for their many ideas and suggestions, and for providing a computer terminal, working space, and an unlimited supply of coffee.

S.P.M.

August, 1982

LIST OF SYMBOLS

CC0(N)	Combinational controllability to logical "0" of node N.
CC1(N)	Combinational controllability to logical "1" of node N.
CCZ(N)	Combinational controllability to high impedance of node N.
CO(N)	Combinational observability of node N.
Dc	Combinational gate "depth" factor.
Ds	Sequential gate "depth" factor.
Mcc	Combinational controllability pseudo-matrix.
Mco	Combinational observability pseudo-matrix.
Msc	Sequential controllability pseudo-matrix.
Mso	Sequential observability pseudo-matrix.
SC0(N)	Sequential controllability to logical "0" of node N.
SC1(N)	Sequential controllability to logical "1" of node N.
SCZ(N)	Sequential controllability to high impedance of node N.
SO(N)	Sequential observability of node N.

TABLE OF CONTENTS

CHAPTER 1. THE TESTING PROBLEM	1.1
CHAPTER 2. VARIOUS APPROACHES TO TESTABILITY	2.1
2.1 Testpoints and Partitioning	2.1
2.2 Empirical Design for Testability Measures	2.2
2.3 Signature Analysis	2.3
2.4 Self-Testing and Fault Tolerant Circuits	2.4
2.5 Level Sensitive Scan Design	2.5
2.6 Testability Analysis	2.8
2.6.1 TMEAS Program	2.9
2.6.2 CAMELOT Program	2.10
2.6.3 TESTSCREEN Program	2.10
2.6.4 SCOAP Program	2.11
CHAPTER 3. SCOAP - VERSION 2.0	3.1
3.1 Controllability/Observability Equations	3.8
3.2 Pseudo-Matrix Equations	3.10
3.3 C/O Calculations	3.10
CHAPTER 4. SCOAP MODELLING CONSIDERATIONS	4.1
4.1 Power Supply Nodes	4.1
4.2 Combinational Cell Modelling	4.2
4.3 Sequential Cell Modelling	4.4
4.4 Treatment of Hierarchical Circuits	4.9
4.5 Bi-Directionality	4.12
CHAPTER 5. ANALYSIS & INTERPRETATION OF RESULTS	5.1
5.1 Floating Output Nodes	5.2
5.2 Fault Modelling	5.2
5.3 Data Reduction	5.4
5.4 Analysis Example	5.11
CHAPTER 6. SUMMARY, CONCLUSIONS & RECOMMENDATIONS	6.1
6.1 Summary	6.1
6.2 Interpretation of SCOAP	6.2
6.3 Conclusions	6.3
6.4 Recommendation for Future Work	6.4
APPENDIX 1. SCOAP EXAMPLE	7.1
APPENDIX 2. BI-DIRECTIONAL EXAMPLES	8.1
GLOSSARY	9.1
BIBLIOGRAPHY	10.1

CHAPTER 1 : THE TESTING PROBLEM

In the design of digital circuits, it is necessary to test the device in order to be certain that it is able to perform the specified function under all of the expected operating conditions. The problems of testing digital circuits are becoming more significant as the costs associated with testing increase. The variety and complexity of testing problems associated with the manufacture and use of digital integrated circuits has been the subject of several books and papers [1-6]. Some of the specific problems of testing are briefly outlined below, and the need for a means of analysing the testability of digital circuits is discussed.

The manufacturer is concerned with the production testing of finished wafers, the testing of packaged devices, and with some sort of Quality Assurance sample testing. The board or system builder must deal with incoming inspection testing, assembled board testing, and system verification. In the field one must be concerned with on-going maintenance and periodic de-bug testing. In some cases, a built in system self-test ("confidence test") is performed on a regular basis to ensure high reliability.

For digital circuits, these tests can be roughly divided into two parts: parametric and functional. Parametric tests are useful for checking the analogue performance of the input receivers and output drivers, the power supply requirements, noise tolerances, and so on. Functional tests are used to verify that the logic circuit functions correctly. They usually consist of a sequence of binary input numbers, where each bit represents the value of a given input, with a corresponding sequence of binary output numbers, where each

bit represents the expected value of a given output. The series of input and output "vectors" form the test sequence for the circuit. For each applied input vector, the actual output values are compared against the expected output vector. Any deviation from the expected values constitutes a failure, and the circuit under test is rejected.

Finding an adequate functional test sequence for large scale digital integrated circuits (LSI or VLSI ICs) is much more difficult than performing parametric tests (with the important exception of mixed analogue-digital ICs, such as switched capacitor filters, and encoder-decoder CODECs, in which the "function" being tested is analogue in nature). It is not practical to design a test which exhaustively sequences through all possible input and internal state conditions for large circuits since the number of steps in the sequence increases exponentially as the number of inputs and state variables increases linearly. A philosophy of "exercising" all nodes is often adopted instead. The input sequence is designed to force all of the gates and nodes in the circuit to change state at least twice, and these changes are (hopefully) detectable at the outputs. It is then assumed that all of the anticipated faults in the circuit will manifest themselves as discrepancies in the output vectors. In fact, if the circuit is not designed properly, it may not be possible to detect a significant portion of these faults, ie. portions of the circuit are untestable. Even if all of the anticipated faults are detectable, it may be impractical to make a test which will completely check for all conceivable faults, since this would require an unreasonable amount of test development, and a very long test sequence. More efficient test patterns, which are based on a functional knowledge of the circuit, can still fail to exercise portions of the logic. The final test sequence is invariably a compromise, designed to uncover as

many of the potential faults in a circuit as possible, given realistic development and testing time constraints.

The test sequence is applied by a special "tester", which determines whether or not the circuit under test is acceptable. The tester is usually a separate piece of equipment, which is expensive to purchase, maintain, and to program. A general purpose digital LSI tester usually consists of a test head and a computer. The test head interfaces the Device Under Test (DUT) to the computer which controls the test. The test head contains the necessary electronics to force and sense voltages and currents on each pin of the DUT, comparators for making pass/fail decisions, and a means of connecting suitable loads or external special function test equipment (eg. spectrum analysers, signal generators, etc.). The test head must be capable of operating at the same speed as the DUT. This will be difficult if the DUT is at the "state-of-the-art", and the tester is of an older (and slower) technology. The test head is often controlled by a dedicated computer, which synchronises the testing, applies the functional test sequences, directs the parametric tests carried out by analogue test circuitry, makes the final sorting decision on each DUT, and performs test data collection and analysis. There is considerable interest now in building parts of the "tester" right into the circuit itself to reduce the hardware demands on the external equipment.

Improvements in fabrication technology have allowed packing densities on ICs to increase to the state where the testing of digital ICs, which was always difficult, has become quite severe. Conventional approaches to testing of ICs have become unworkable. Traditionally, the IC test, particularly the test pattern which determines the functionality of the circuit, was developed quite late in the design process. In large and

complex designs testing was usually enhanced by the addition of test points at strategic nodes in the circuit, or by partitioning the circuit into sub-circuits which could be tested independently of each other. For large random logic circuits with many sequential states and several internal feedback paths the choice of test point locations and partitioning boundaries is not always obvious. For such circuits some way of predicting the testing difficulty in advance would be very desirable. It would also be very beneficial to know, in advance, whether any measures taken to improve the inherent "testability" will actually be effective.

For small, uncomplicated logic circuits it is possible to manually perform some sort of "testability analysis", ie. to quickly assess the length of the test sequence required and spot nodes which no test pattern will adequately test, without actually writing a test sequence. For large circuits this task is not trivial, and a manual inspection is likely to overlook some non-obvious problem areas, and raise false alarms elsewhere. However, if the testability analysis task can be described algorithmically, there is no reason why it cannot be automated for greater speed and reliability. Logic simulators (eg. TEGAS, F/LOGIC [7,8]) are available to determine whether the logic designer's circuit actually accomplishes the same function as the one described in the original specification. Fault simulators (eg. TEGAS and F/LOGIC again) are available to determine the effectiveness of the testing engineer's test sequences in detecting faults in the logic. Some fault simulators (eg. TEGAS) can be used to generate test sequences automatically, using random or deterministic means, but they are not (yet) practical for general case VLSI designs. Recently a number of software

aids (eg. SCOAP, TESTSCREEN, TMEAS, CAMELOT, [9-17]) have also become available to attempt to analyse the testability of a circuit before the test pattern itself is generated. Also, a number of design methodologies have been developed to make circuits inherently more testable, the most notable being Level Sensitive Scan Design (LSSD), and self testing ICs [18-21]. Design For Testability (DFT) rules and algorithms have been available for some time, and there are a number of good sources for such procedures [2,3,6,22].

By itself, testability analysis is only useful in determining the ease or difficulty in testing a given circuit. It is much more powerful when used in conjunction with a design methodology, or even with a handful of ad-hoc rules and guidelines. The testability analysis tool allows one to see the effectiveness of whatever techniques are being applied to the testing problem.

A design strategy, which uses logic simulation, fault simulation, and testability analysis together, is illustrated in Fig. 1.1. From the functional specification a circuit design is produced which is then subjected to a functional verification (using a logic simulator). If the function performed by the circuit is not the same as the one described in the specifications the circuit is re-designed.

A testability analysis is then performed in order to determine if the design is testable, before any effort is expended trying to generate a test sequence. If the circuit is considered to be untestable, or unacceptably difficult or expensive to test, then it will have to be modified. A new functional verification is required to be sure that the new design still implements the intended function, followed by a new evaluation to see if the modifications were effective in improving the testability.

Consider the uni-directional bus node, B, with tri-state drivers and receivers, as in Fig. 4.8. There are n inputs, X_1 to X_n , which are enabled by n control lines EX_1 to EX_n . There are also m outputs, Y_1 to Y_m , enabled by m control lines EY_1 to EY_m respectively.

Using the present SCOAP algorithm, one could calculate the controllabilities for X_1 to X_n , EX_1 to EX_n , and for EY_1 to EY_m , plus the observabilities for Y_1 to Y_m . It is assumed that while any number of receivers may be enabled, only one driver at a time is enabled. The junction between the drivers and receivers would normally be treated as a Wired-OR connection, as mentioned earlier (see Fig. 4.8a). This artificially adds n nodes to the circuit, because the Wired-OR connection is not actually a gate. Also, this method does not account for the need to guarantee that one and only one driver at a time is enabled.

Rather than adding the artificial Wired-OR gate, an alternative method is to instead treat node B as if it were a special logic element with $(n+m)$ ports, which can be inputs or outputs at different times. If drivers and receivers are paired, which is often the case in a bus-oriented design, then these bi-directional node logic elements are on the boundaries between the various blocks of unidirectional logic, as in Fig. 4.9.

Controlling node B in the alternative method involves not only propagating the desired value to it via the uni-directional logic, but also includes enabling one and only one driver and disabling all of the others, that is

Once the testability is judged to have reached a satisfactory level, a test sequence is generated and modelled (via a fault simulator) to see how effective it is for detecting faults in the circuit. The test sequence is evaluated and modified until the "coverage" (ie. the percentage of the total set of faults being modelled that are detected by the sequence) is acceptable.

What constitutes an "acceptable" level of testability or fault coverage is often a very qualitative, rather than quantitative, measure. The results of the fault simulations for many circuits of different complexity and test sequences of different lengths and coverage are used to "tune" the testability evaluation process.

It is essential to be aware that all circuit designers are under considerable pressure to produce a product as soon as is physically possible. This is especially true if early completion of the design phase will give the manufacturer a significant advantage in marketing the IC itself, or other products in which it is a vital component, over similar devices being offered by competitors. Understandably, the designer will not be inclined to apply DFT techniques or to perform a testability analysis on a new IC if these techniques significantly increase the cost and complexity of the design, if valuable time is lost in the process, or if the benefits are not immediately obvious.

The nature of the digital IC testing problem has been discussed above. In the following chapters various approaches to the problem are considered, with special emphasis on automated testability analysis. The rationale, algorithms,

and limitations of the SCOAP program for testability analysis are detailed, with several suggested improvements. Problems in the analysis and interpretation of results are pointed out, with some recommendations for resolving them. Appendices are also included which contain examples to illustrate the present SCOAP algorithms and some of the suggested changes.

The SCOAP testability analysis program, Version 2.0, was made available to the author by the University of California (Berkeley) in the spring of 1981, when it was installed on a VAX 11/780 at Mitel Corporation in Kanata. The program was newly released, and largely untested. Several small programming errors were detected, fixed, and reported to UCB, which is responsible for maintaining the program and seeing that the updated versions which are made available to users are consistent. In actual fact, an informal "users' group" was formed between UCB, the author, and some other individuals at private companies who were attempting to see if the SCOAP program could be useful in alleviating some of their digital circuit testing problems.

A special SCOAP cell library (which contains the testability information for each type of gate used in digital circuits) was generated by the author for the Mitel 50000 series Uncommitted Logic Array family, using the cell library which was originally provided with the program as a starting point. A translator program was also written by the author to generate SCOAP compatible input files from the gate-level network description files for logic circuits. Another program was written by the author which condenses the data from the SCOAP output files and produces a list of nodes which are untestable according to SCOAP, a list of the remaining nodes sorted by testability, and a histogram of the testability

distribution for each circuit. The most difficult problem however, was attempting to analyze the output of the SCOAP program. The documentation, which accompanied the program explained how to run the software, but not how to interpret the results.

By means of circuit examples, the author has been able to study and experiment with the SCOAP program in some detail in order to understand the assumptions, algorithm, and the limitations of this particular testability analysis routine.

CHAPTER 2 : VARIOUS APPROACHES TO TESTABILITY

A number of different approaches to improving the testability of digital circuits are briefly discussed below, each of which has advantages, disadvantages, and specific applications.

2.1 : Testpoints and Partitioning

Two of the most basic approaches to improving the testability of a digital circuit are: (a) to provide readily accessible nodes in the circuit solely for the purpose of testing, and (b) to divide the circuit into sections which can be tested independently of each other. The first method is the addition of testpoints, the second is known as partitioning.

The purpose of adding testpoints is to gain access to nodes which are logically very far removed from the inputs and outputs of the circuit. These points can be simply monitored as if they were extra outputs, or can be used to inject data into the circuit, bypassing a great deal of circuitry and thereby speeding up and simplifying the functional test. In circuit board testing, testpoints can physically be located anywhere on the surface of the board. In IC manufacturing, the entire surface of the chip is covered with a thick, protective oxide or nitride layer, except for the bonding pads on the perimeter. In IC design, the number of bonding pads and package pins available to be assigned is limited, and the maximum possible use is made of each one, to the point of multiplexing functions if the packing density requires it.

Partitioning is not difficult if the circuit is composed of two or more sections which are easily disconnected and then tested separately. If there are several feedback paths in

the whole circuit between otherwise separate function blocks, testing will be much simpler if the feedback paths can be opened during the test, then reconnected for normal operation. In some designs the feedback loops are too numerous and too interdependent to be removed. Since the boundaries of the function blocks are not clearly defined, partitioning is a much more difficult task.

2.2 : Empirical Design For Testability (DFT) Measures

There is a whole body of knowledge based on common sense and years of experience for simplifying the task of testing circuits. Most of these empirically derived techniques are more applicable to board level testing than to IC testing, and many are related to the problems experienced by technicians servicing equipment "on site" or "in the field". Some of these methods are:

1. Testpoints and partitioning as in section 2.1 above.
2. Additional Enable/Disable logic which isolates portions of the circuit under test from each other, achieving partitioning by electrical means.
3. Disabling the normal internal clock signals and applying external clocking for testing which causes the circuit to operate at a faster rate for enhanced testing speed.
4. Circuit initialization or reset of internal states to a known starting condition via an input or testpoints.
5. Resistor pull-ups and pull-downs instead of direct connections to allow the inputs to gates (on boards) to be manually connected to either voltage level for testing.
6. Design "tricks" which save devices or space, but make the circuit harder to test and verify, are avoided.

Note that there is a conflict raised by point 6 above between the designer's aim of maximizing efficiency and minimizing space and complexity, and the test engineer's desire to have easily testable logic. In general, DFT rules and guidelines

require additional circuitry: more devices, more pins, and more space. None of them suggest circuit simplifications. Also, in dealing with production testing of ICs, many of these measures are not applicable since one cannot probe into the interior of a packaged chip. However, these methods are still valid for board and system level testing and they are very thoroughly described in sources such as [2,3,6,22].

2.3 : Signature Analysis

For highly sequential circuits, a long test sequence is entered into the inputs, and the output vectors checked against a set of pre-determined expected values. The test sequence is chosen such that most, if not all, of the logic is exercised and the probability that single and multiple faults in the circuit will not be detected is small. If a very long test sequence is desirable or necessary, the test time could be very long, especially if the speed is limited by the tester, not the device under test. Rather than examining each output vector in the test sequence, it may be feasible to run the device through a long test sequence, and compress the output vectors into a single binary number. The binary number which results from this compression is called a "signature".

For the desired test sequence the signature of a fault free circuit is not unique. Some faults in the circuit will produce the same signature, but the test can be designed so that the probability of a faulty device producing the "pass" signature is very small. Typically, the test sequence is generated pseudo-randomly, and the expected signature is determined by running the circuit under test and the data compression circuit on either a circuit simulator or a working prototype system.

For debugging at the board and system level, certain frequently occurring faults have their own characteristic

signatures. It is possible to use this in "guided probe" testing, where the testing system tells the service technician which part of the circuit has failed, or which part is to be tested next. While the freedom to probe anywhere at will does not exist within ICs, signature analysis techniques can be incorporated into self-testing or partially self-testing digital ICs to improve their testability. It is possible to store the input and expected output vectors (in an on-chip ROM perhaps), and build in the circuitry required to perform the test. There is the advantage of being able to perform the test at the normal operating speed of the device, rather than be limited by the performance of an external tester.

In summary, the advantages of signature analysis are:

1. The tester hardware requirements are minimal.
2. Minimal storage of output test vectors.
3. The input vectors can be self-generated.
4. The test can be quickly generated using a working system.
5. The test runs at the normal operating speed of the device.

2.4 : Self-Testing and Fault Tolerant Circuits

Self-testing Circuits (STC) are able to detect certain internal faults, and are able to signal that a fault has occurred. Since STC designs perform the testing internally, the demands on the external testing hardware, software, and the skill and training of the test or service engineers are reduced.

If the circuit is also able to correct the fault before an error occurs in the output, then the circuit is said to be "fault tolerant". Fault Tolerant Circuits (FTC) are much harder to test because they inherently disguise faults which do not result in output errors. In testing a FTC it is difficult to know whether the entire circuit is working or if

the circuit is working with the maximum number of faults allowable without producing errors. Fault tolerant techniques are useful in building very reliable systems from tested components of known reliability. For testing purposes it is preferable to have automatic fault detection, rather than fault correction.

STC designs may contain a subcircuit called a "checker" which could simply be a circuit which detects when an output is generated that is not in the allowed set of responses. The checker, in that case, would fail to detect faults which produced allowed, but incorrect, output responses. A more complete design can incorporate a miniature tester which either generates the whole test sequence for the device, or performs a signature analysis on the output after a given number of cycles.

A recent development, the Built-In Logic Block Observation (BILBO) technique, uses registers which are capable of performing signature analysis type tests on blocks of combinational logic [20]. Data can also be read into or out of the BILBO registers in the normal "system" mode. This technique is very similar to the IBM Level Sensitive Scan Design (LSSD) which follows.

2.5 : Level Sensitive Scan Design (LSSD)

Eichenberger and Williams of IBM described a novel approach [18] which was intended to solve two problems: eliminate hazard (race) conditions, and simplify testing. The first problem was solved by designing all sequential gates (latches) so that they were not dependent on signal rise, fall or delay times. Also some design rules were stated to prevent possible hazards (eg. clock signals may not be ANDed together, and clock signals may only be connected to the clock inputs of latches). The second problem was overcome by using a special latch design, called a "Shift Register Latch", or SRL.

SRL block

SRL schematic

Serially connected
SRL's on an IC Serially connected
SRL's in a system

Fig. 2.1 : Shift Register Latch (SRL) & LSSD circuits [18]

This design, shown in Fig. 2.1a & b, has two halves, as in a Master-Slave flip/flop. The upper section, L1, behaves as a normal sequential storage element in the circuit. During normal operation, it is clocked via the system clock, C, and transfers data from input D to output L1. During testing, the system clock, C, is disabled, and the two testing clocks, A and B, are activated. Data is then transferred from input I to output L2.

For testing, the L2 latches of all SRL's in the circuit are connected together in a long shift register, as illustrated in Fig. 2.1c. With the system clock disabled, and the testing clock running, the values of all of the L1 outputs at the instant the clock was frozen can be read out serially from test pin OUT. Not only can the "state" of a sequential circuit be read out serially, but the circuit latches can be initialized by pumping a serial bit sequence into the IN pin with the testing clock running. Pins IN and OUT of all the ICs on a PC board can be connected the same way, as in Fig. 2.1d. The main advantage of this approach is that the sequential testing problem has been reduced to a combinational one. Sequential gates which were virtually inaccessible using conventional testing techniques become directly accessible via the serial shift register.

One to three extra pins have to be dedicated to testing (IN, OUT, and TEST), which is not in itself a major drawback, considering the benefits which are possible. However, the main disadvantage of this approach is that it increases the layout area and interconnect by 5% to 40%, which may be unacceptable. If economics dictates that only partial LSSD must be used, then testability analysis is useful to determine which latches should be included in the SRL shift register chain to achieve the optimum testability.

2.6 : Testability Analysis

The problem with applying most of the above DFT techniques is that one is unsure of the effectiveness of the measures being taken. Testability analysis (TA) gives an indication of the difficulty in testing a circuit design, and can be used to predict the relative improvement in testability if some DFT measures were to be implemented. For small circuits, one can manually detect potential testing problems by looking for nodes which are either inaccessible or difficult to control or to observe. For large circuits the problem areas may not be readily obvious, so it is desirable to automate the analysis process. The goals of performing an automated testability analysis are:

1. To predict the length of the test sequence required in order to test the circuit. This is directly related to the cost of testing the circuit.
2. To be able to detect and identify any portion of the circuit which may be untestable, either because it is inaccessible, it cannot be initialized, or it is redundant.
3. To be able to indicate the effectiveness of using design for testability techniques, whether ad-hoc methods or a more structured approach is used.

There are several different approaches, with differing algorithms, but several terms extensively used in testability analysis are common to all of them.

The "primary inputs" are the nodes of a digital circuit which are directly accessible for applying input test vectors. The "primary outputs" are those nodes which are directly accessible for reading the output vectors for the pass/fail comparison. These nodes are the inputs and outputs which connect the circuit under test to other components of the system (eg. edge connector pads of boards, and pins of packaged ICs). Testpoints may be treated as primary inputs and outputs on

boards, but the internal nodes of ICs are not directly accessible to probing, and they are neither primary inputs nor outputs.

The "controllability" of a given node in a digital circuit represents the degree of difficulty in accessing and stimulating that node via the primary inputs. The "observability" of a given node represents the degree of difficulty in accessing and observing the logical value of that node via one of the primary outputs. How this "degree of difficulty" is assessed depends on the method used in performing the testability analysis. For both the controllability and observability it is necessary to create a path in the logic through which the desired signals can propagate. This process is called "path sensitization". It is preferable to sensitize a path for the signals which requires the fewest possible number of test sequence steps, and involves the smallest number of gates and nodes possible, that is the most direct route. There are several different approaches, with differing algorithms, and three of these are considered below.

2.6.1 : TMEAS Program

TMEAS stands for Testability MEASure, a program by Grason of Bell Labs [13,14] which analyses the testability of a circuit by making use of transfer functions. The controllability of the outputs is expressed as a function of that for the inputs. The observability of the inputs is a function of that for the outputs. Once a transfer function for a logic block is derived, the function can be used in a larger system in which the logic block is a submodule. This can be extended from the gate level right up to the system level. The hierarchical capability of TMEAS makes it very suitable to the testing of boards and systems in which the submodules are ICs.

2.6.2 : CAMELOT Program

CAMELOT, which stands for Computer Aided MEasure for Logic Testability, was developed by Bennetts, Maunder, and Robinson for the British Post Office [15,16]. It is very similar to TMEAS, except for the way in which the observabilities of input nodes to modules are calculated. TMEAS assumes that the observability of a input node of a module will be independent of the controllabilities of the other inputs. CAMELOT allows for dependencies of the observabilities on the controllabilities of what are called "supporting inputs".

2.6.3 : TESTSCREEN Program

TESTSCREEN is a testability analysis program written by Kovijsanic at Sperry Research [11]. Unlike TMEAS, it does not use transfer functions to calculate the testability of nodes in a circuit. It is not hierarchical, and does not easily lend itself to analysing large systems. However, it does estimate the number of sequence steps, and the degree of difficulty in testing nodes in ICs. Six testability figures are calculated for each node, and all are integers. Once the program has calculated the testability figures, it is necessary for the test engineer to interpret them. In general, high scores indicate difficult-to-test nodes.

Kovijsanic has attempted to alleviate the analysis problem by calculating a single "Figure of Merit" [12] for each circuit being analysed. The Figure of Merit is based on the testability scores, the number of nodes and cells (gates) in the circuit, an estimate of the number of feedback loops in the circuit, and the number of input and output pins on the device. This number is an overall indication of the general testability of the entire circuit. To pinpoint problem areas, or to see the effectiveness of specific DFT techniques, one must examine the testability scores in detail.

2.6.4 : SCOAP Program

SCOAP, which stands for Sandia Controllability/Observability Analysis Program, is a testability analysis program written by Goldstein [9,10] which is very similar to TESTSCREEN above. It calculates six integer testability scores for each node in the circuit, where an increase in a given score reflects an increase in the difficulty in testing that particular node. Because of the large amount of data that is produced, the same analysis and interpretation problem exists for large circuits as with TESTSCREEN. The main differences are in the assumptions which are made.

SCOAP assumes that all of the inputs to each cell (gate) are totally independent of each other, and takes the sum of the scores for controlling the inputs necessary to achieve the desired output response. For some gates TESTSCREEN assumes that in controlling the most difficult to control input, you have also been able to control the easier ones. The maximum score is taken, not the sum. As a result TESTSCREEN scores are generally more optimistic than SCOAP.

SCOAP attaches a "depth factor" to combinational gates to indicate the difficulty in testing nodes via these gates. For example, it is considered to be easier (and preferable) to test a certain node via a path which went through only four gates than one which went through six. The depth factor allows one to make the distinction. (TESTSCREEN does not make this distinction, since for combinational circuits depth affects propagation delay but not the output value. Both programs have a depth factor for sequential gates, which represents clock cycles, and accounts for the number of sequence steps needed to access a given node.)

In SCOAP, the "controllability" of a node is related to the minimum number of nodes, or clock cycles, which must be assigned values in order to obtain a logical "0" or "1" at the node in question, working from the primary inputs. The observability of a node is related to the minimum number of node assignments, or clock cycles, necessary to propagate the value of the node to a primary output.

For example, in Fig. 2.2a, fifteen node assignments are required to control node N to a logical "1" (eight primary inputs and seven internal gates), whereas only four node assignments (any primary input and three internal gates) to set N to logical "0". In Fig. 2.2b, node M can be observed either via primary output A or output B. It is easier to observe M via A (only two primary input node assignments and two internal gates, versus three primary inputs and three internal gates for B), and this is the path that would be chosen by SCOAP.

The SCOAP algorithm is explained in greater detail in the following chapter for combinational and sequential circuits.

- a) Controllability : To obtain a "1" at node N, 8 primary inputs and 7 internal gates must be assigned values.

To obtain a "0" at node N, any 1 primary input and 3 internal gates must be assigned values.

- b) Observability : To observe the value of M via primary output A, one must go through 2 gates, and control 2 primary inputs. To observe M via output B, one must go through 3 gates, and control 3 primary inputs.

Fig. 2.2 : SCOAP (Combinational) Controllability/Observability

CHAPTER 3 : SCOAP VERSION 2.0

The SCOAP program analyses the testability of a given circuit by estimating the ease of stimulating and observing each node in the circuit from the primary input and output nodes. What follows is an interpretation of the original SCOAP description by Goldstien [9,10].

SCOAP characterizes the testability of a given node (N) with six functions:

CC0(N) - Combinational Controllability of N to logical "0"
 CC1(N) - Combinational Controllability of N to logical "1"
 CO(N) - Combinational Observability of N
 SC0(N) - Sequential Controllability of N to logical "0"
 SC1(N) - Sequential Controllability of N to logical "1"
 SO(N) - Sequential Observability of N

These functions have integer values which are intended to provide a quantitative measure of the difficulty of controlling and observing the logical values of the internal nodes of the circuit. The functions are not based on any particular testing philosophy or methodology, except to minimize the testing cost. In general, the higher the integer value, the more difficult and the more expensive it will be to test a particular node.

The Combinational Controllability is a function of the minimum number of gates and nodes which must be assigned values in order to obtain a logical zero or one at a given node. The Combinational Observability is a function of the minimum number of gates and nodes between the given node and a primary output, plus the minimum number of nodes which must be assigned values in order to propagate the logical value of the given node to a primary output. The Sequential Controllability is a function of the minimum number of clock cycles necessary to obtain the desired value at a given node.

The Sequential Observability is a function of the minimum number of clock cycles necessary to propagate the signal at the given node to a primary output.

Two examples of how the Combinationality/Observability (C/O) functions are derived and used are given: a 2-input OR gate (Fig. 3.1a), and a positive-edge triggered D flip-flop (Fig. 3.2a). The 2-input OR gate output can only be set to "0" if both inputs are low, but can be set to "1" if either input is high. Note that the option of setting both inputs high to set the output to "1" is not included in the equations in Fig. 3.1b, since this would result in a higher score than the case where only one input is high. Also note that to observe the logical value at one of the inputs via the output, it is necessary to set the other input to "0", in order to allow the desired value to propagate through the gate. The C/O relations are expressed in their most concise form in the pseudo-matrix equations of Fig. 3.1c (the form in which they are stored in the SCOAP cell library). These relations are not true matrix equations in the formal sense. While the number of columns in the C/O pseudo-matrices is fixed, depending only on the number of inputs and outputs of the gate being modelled, the number of rows varies, depending on the number of optional ways of calculating the controllability or observability. The matrix type of notation is only a convenient way of expressing the relationship between controllabilities and observabilities of inputs and outputs.

The D flip/flop in Fig. 3.2a requires a positive edge to propagate the signal from D to Q. Therefore, it is necessary to provide both a "0" and a "1" on the clock input CK in the equations of Fig. 3.2b in order to control Q to the desired value, and also to observe D via Q. Note that in order to observe the clock, it is necessary to change the state of Q from "0" to "1" or vice versa, and that the clock itself must be toggled in order to be observed. What is being observed

OR Gate : $C = A + B$

$$CC0(C) = CC0(A) + CC0(B) + 1$$

$$CC1(C) = \min(CC1(A)+1, CC1(B)+1)$$

$$CO(A) = CO(C) + CC0(B) + 1$$

$$CO(B) = CO(C) + CC0(A) + 1$$

$$SC0(C) = SC0(A) + SC0(B)$$

$$SC1(C) = \min(SC1(A), SC1(B))$$

$$SO(A) = SO(C) + SC0(B)$$

$$SO(B) = SO(C) + SC0(A)$$

(a)

(b)

$$\begin{bmatrix} CC0(C) \\ CC1(C) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ \min\{0 & 1 & 0 & 0 & 1\} \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} CC0(A) \\ CC1(A) \\ CC0(B) \\ CC1(B) \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} CO(A) \\ CO(B) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} CO(C) \\ CC0(C) \\ CC1(C) \\ CC0(A) \\ CC1(A) \\ CC0(B) \\ CC1(B) \\ 1 \end{bmatrix}$$

(c)

$$\begin{bmatrix} SC0(C) \\ SC1(C) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ \min\{0 & 1 & 0 & 0 & 0\} \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} SC0(A) \\ SC1(A) \\ SC0(B) \\ SC1(B) \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} SO(A) \\ SO(B) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} SO(C) \\ SC0(C) \\ SC1(C) \\ SC0(A) \\ SC1(A) \\ SC0(B) \\ SC1(B) \\ 1 \end{bmatrix}$$

Fig. 3.1 : Two-input OR gate

D-flip/flop, positive edge clocked

$$CC0(Q) = CC0(D) + CC0(CK) + CC1(CK)$$

$$CC1(Q) = CC1(D) + CC0(CK) + CC1(CK)$$

$$CO(D) = CO(Q) + CC0(CK) + CC1(CK)$$

$$CO(CK) = \min(CO(Q) + CC0(Q) + CC1(D) + CC0(CK) + CC1(CK), \\ CO(Q) + CC1(Q) + CC0(D) + CC0(CK) + CC1(CK))$$

$$SC0(Q) = SC0(D) + SC0(CK) + SC1(CK) + 1$$

$$SC1(Q) = SC1(D) + SC0(CK) + SC1(CK) + 1$$

$$SO(D) = SO(Q) + SC0(CK) + SC1(CK) + 1$$

$$SO(CK) = \min(SO(Q) + SC0(Q) + SC1(D) + SC0(CK) + SC1(CK) + 1, \\ SO(Q) + SC1(Q) + SC0(D) + SC0(CK) + SC1(CK) + 1)$$

(a)

(b)

$$\begin{bmatrix} CC0(Q) \\ CC1(Q) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} CC0(D) \\ CC1(D) \\ CC0(CK) \\ CC1(CK) \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} CO(D) \\ CO(CK) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ \min\{1 & 1 & 0 & 0 & 1 & 1 & 1 & 0\} \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} CO(Q) \\ CC0(Q) \\ CC1(Q) \\ CC0(D) \\ CC1(D) \\ CC0(CK) \\ CC1(CK) \\ 1 \end{bmatrix}$$

(c)

$$\begin{bmatrix} SC0(Q) \\ SC1(Q) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} SC0(D) \\ SC1(D) \\ SC0(CK) \\ SC1(CK) \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} SO(D) \\ SO(CK) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ \min\{1 & 1 & 0 & 0 & 1 & 1 & 1 & 1\} \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} SO(Q) \\ SC0(Q) \\ SC1(Q) \\ SC0(D) \\ SC1(D) \\ SC0(CK) \\ SC1(CK) \\ 1 \end{bmatrix}$$

Fig. 3.2 : D-type flip/flop (positive edge clocked)

there is that the clock signal toggles, not its particular value at a given instance. Note also that the D flip/flop is a sequential gate, unlike the 2-input OR gate above, and this is reflected in both the equations and the arrays where a sequential depth factor of unity is added. (However, not all types of sequential gates can be modelled, as will be demonstrated in a later section).

In the SCOAP library each type of logic gate or cell has four characteristic pseudo-matrices which formally express the controllabilities and observabilities of the outputs in terms of those for the inputs, and vice versa. Each element of these pseudo-matrices is a binary number, of value "1" or "0", which indicates whether or not a particular controllability or observability term is a factor in the testability function being calculated.

The matrix notation is used here as a matter of convenience, a way of showing the relation between the input and output controllabilities and observabilities. These relations are not linear (as might be implied by the use of the matrix notation) because of the presence of "minimum" operators in between some of the rows of an array. The pseudo-matrix array format is useful in understanding the manner in which the testability information is stored in the SCOAP cell library.

When, for instance, a node connected to the 2-input OR gate is encountered in a testability analysis problem, the pseudo-matrices are called from a cell library. The C/O figures for this node are calculated, using the available C/O data for the other nodes also connected to the same OR gate. If the new C/O figures are lower for the node, then the old C/O data is updated.

Consider a generalized gate with input nodes X_1 to X_n , and output nodes Y_1 to Y_m , as in Fig. 3.3. There are four equations for calculating the controllability of each output from those of

$$CC_o = M_{cc} CC_i$$

$$CO_i = M_{co} CCO_{io}$$

$$SC_o = M_{sc} SC_i$$

$$SO_i = M_{so} SCO_{io}$$

NOTE: The matrix notation shown here and in Fig. 3.4 is used to show the relationship between input and output controllabilities and observabilities. Because of the presence of minimum (M) operators, the relationships are all non-linear. The pseudo--matrices above may be regarded as "array operators" which define the relationships between the input and output vectors. This notation is not essential to the understanding of the SCOAP program.

Fig. 3.3 : Generalized gate and C/O pseudo-matrix notation

Combinational Controllability

$$\begin{bmatrix} CO(Y1) \\ CI(Y1) \\ CO(Ym) \\ CI(Ym) \end{bmatrix} = Mcc \begin{bmatrix} CO(X1) \\ CI(X1) \\ CO(Xn) \\ CI(Xn) \\ 1 \end{bmatrix}$$

Combinational Observability

$$\begin{bmatrix} CO(X1) \\ CO(Xn) \end{bmatrix} = Mco \begin{bmatrix} CO(Y1) \\ CI(Y1) \\ CO(Ym) \\ CI(Ym) \\ CO(X1) \\ CI(X1) \\ CO(Xn) \\ CI(Xn) \\ 1 \end{bmatrix}$$

)-previous state data

Sequential Controllability

$$\begin{bmatrix} SCO(Y1) \\ SCI(Y1) \\ SCO(Ym) \\ SCI(Ym) \end{bmatrix} = Msc \begin{bmatrix} SCO(X1) \\ SCI(X1) \\ SCO(Xn) \\ SCI(Xn) \\ 1 \end{bmatrix}$$

Sequential Observability

$$\begin{bmatrix} SO(X1) \\ SO(Xn) \end{bmatrix} = Mso \begin{bmatrix} SO(Y1) \\ CI(Y1) \\ SCO(Ym) \\ SCI(Ym) \\ SCO(X1) \\ SCI(X1) \\ SCO(Xn) \\ SCI(Xn) \\ 1 \end{bmatrix}$$

)-previous state data

Fig. 3.4 : SCOAP C/O matrix notation for a general gate

(N.B. : See also the note on Fig. 3.3.)

the inputs, and two equations for calculating the observability of each input from the other testability functions. For the gate of Fig. 3.3, these equations can be grouped together into the pseudo-matrix equations in Fig. 3.4. In order to clarify the SCOAP algorithms, the equations themselves are presented more formally below.

3.1 : Controllability/Observability Equations

The Combinational Controllabilities of output Y_i are:

$$CC0(Y_i) = f_{cc0}(CC0(X_1), CC1(X_1), \dots, CC0(X_n), CC1(X_n), D_c) \quad (3.1)$$

$$CC1(Y_i) = f_{cc1}(CC0(X_1), CC1(X_1), \dots, CC0(X_n), CC1(X_n), D_c) \quad (3.2)$$

where the combinational controllabilities are as described earlier and D_c is the combinational depth factor assigned to the gate. The functions f_{cc0} and f_{cc1} depend on the logic function of the cell or gate being modelled. Whether or not each combinational controllability is included in the calculation depends on whether or not they are necessary to control the output to the desired state. Whether or not the depth factor, D_c , is included depends on whether the function is combinational or sequential.

The Combinational Observability of input X_j of the same gate is:

$$CO(X_j) = \min_{i=1,m} (CO(Y_i) + f_{coi}(CC0(Y_i), CC1(Y_i), CC0(X_1), CC1(X_1), \dots, CC0(X_n), CC1(X_n), D_c)) \quad (3.3)$$

where the depth factor and the combinational controllabilities and observabilities are as defined earlier. Note that if X_j can be observed via any of the m outputs, Y_i , then the minimum is taken of the m possible values. If there are different ways of observing X_j via each output, Y_i , then the total number of possibilities can be much greater than m .

Given the same gate with input nodes X_1 to X_n , and output nodes Y_1 to Y_m , the Sequential Controllabilities of output Y_i are similarly defined:

$$SC_0(Y_i) = f_{sc_0}(SC_0(X_1), SC_1(X_1), \dots, SC_0(X_n), SC_1(X_n), D_s) \quad (3.4)$$

and:

$$SC_1(Y_i) = f_{sc_1}(SC_0(X_1), SC_1(X_1), \dots, SC_0(X_n), SC_1(X_n), D_s) \quad (3.5)$$

where the sequential controllabilities are as defined earlier and D_s is the sequential depth factor assigned to the gate.

The Sequential Observability of input X_j of the same gate is:

$$SO(X_j) = \min_{i=1, m} (SO(Y_i) + f_{so_i}(SC_0(Y_i), SC_1(Y_i), SC_0(X_1), SC_1(X_1), \dots, SC_0(X_n), SC_1(X_n), D_s)) \quad (3.6)$$

where the sequential controllabilities and observabilities are as defined earlier.

The depth factors D_c and D_s in the above relations are used to account for the presence of the gate in the sensitization path. In SCOAP the value of the depth factor is set to unity for either the combinational or the sequential, but not both since a gate is of either one type or the other. Whether or not unity is the optimum value is arguable. For complex sequential gates the depth factor may actually be greater if multiple clock cycles are necessary to propagate a signal from input to output. For a purely combinational circuit, the circuit depth affects propagation delay, but not final output value, so the depth factor is less meaningful.

3.2 : Pseudo-Matrix Equations

As mentioned earlier, the above equations can be expressed more concisely in a pseudo-matrix equation notation:

$$\begin{aligned} \text{CCo} &= \text{Mcc CCI} , & \text{COi} &= \text{Mco CCOio} , \\ \text{SCo} &= \text{Msc SCI} , & \text{SOi} &= \text{Mso SCOio} . \end{aligned} \quad (3.7)$$

where Mcc, Mco, Msc, and Mso are the characteristic pseudo-matrices and the other terms are the corresponding vectors in Fig. 3.4. The pseudo-matrices Mcc, Mco, Msc, and Mso consist of rows of binary numbers. The product of a row of the M matrix and the column sums up the necessary terms for the calculation (the others become zero and drop out). Because there is often more than one combination of terms which describe the function, it is often necessary to take more than one row-column product, and then take the minimum of these products. For these cases, two or more binary rows are used for a particular calculation. (In the gate library file, such rows are separated by an indicator which tells the program to take the minimum product of the row immediately above and below with the column vector.)

3.3 : C/O Calculations

When an entire circuit is being considered, all of the C/O values are set to infinity (in the program, an arbitrarily large integer, 99999, is used to represent infinity), except for:

1. Primary input nodes, whose controllability values are set to unity in the combinational case and to zero in the sequential case. To set a primary input node to the desired value only requires that one node, namely itself, be controlled directly by the tester.
2. Primary output nodes, whose observability values are set to zero. This is because the value of a primary output can be read at any time without having to sensitize an internal path through the circuit.

The controllability values are calculated for the internal nodes of the circuit, starting from the primary inputs, working towards the primary outputs, until all nodes have been considered. In practice, this involves several iterations through the entire circuit until all controllability values have settled. In order to prevent instability in the iteration process, individual values are only allowed to decrease. If a value for a node is calculated in two or more equations, because that node fans out to several gates, the minimum value is taken and the others discarded.

Once the controllability values have stabilized, the observabilities are calculated, working from the primary outputs back towards the primary inputs. Again, several iterations will be required through the entire circuit, and values are allowed only to decrease in order to avoid oscillations.

The program is designed to analyse a given circuit gate by gate, and it examines the nodes which connect to the gate one at a time. It is assumed that all of the nodes around a gate are independent of each other. No attempt is made to account for the fact that nodes may be inter-related. For example, it is possible that the optimum condition of the inputs for controlling an output to a desired value is not possible because the inputs are not independent. Or, in controlling one particular input to a certain value, another input is already set to its required value. In that case, by controlling the most inaccessible input, most or all of the difficulty in controlling the gate has been overcome. Because the algorithm is designed to examine a circuit gate by gate, it does not account for such dependencies. As a result, the C/O values are only estimates based on the assumption of being able to consider each gate locally and separate from the network surrounding it.

At the end of the calculations, there are six final C/O values for each node in the circuit. Nodes which SCOAP finds to be uncontrollable or unobservable retain the initial value of infinity (ie. 99999), and the others will have lower values, depending on the testability.

CHAPTER 4 : SCOAP MODELLING CONSIDERATIONS

There are some important considerations which must be kept in mind when generating SCOAP input data, which describes a given circuit to the program, and the cell library, which contains all of the testability information for all of the different types of gates which are found in that circuit:

1. Initialization of power supply nodes.
2. Modelling of combinational gates or cells.
3. Modelling of sequential gates or cells.
4. Treatment of hierarchical circuits.
5. Treatment of bi-directional nodes.

4.1 Power Supply Nodes

Unused inputs of gates are tied to logical "0" or "1" in order to reduce the logic function to one involving only the other inputs. These values are usually fixed at the power supply voltages, and do not change while the circuit is in operation. These input nodes cannot be modelled in the same way as the other nodes in the circuit, which are free to change value as necessary. The "fixed-value" node is perfectly controllable to the intended value, and it is totally uncontrollable to the other. For example, a "Ground" signal has CC0 of unity, and CC1 of infinity. A gate which has the ground connected to one of the inputs may be controllable to any state where a "0" is necessary on the ground (depending on the controllabilities of the other inputs). If a "1" is necessary on the ground to set the gate to a particular state, that state will turn out to be uncontrollable (ie. unattainable).

For a custom IC design, one normally includes only as many inputs to a gate as are actually needed. In board design, and in standardized cell library IC designs, there is a need to be able to model fixed-value nodes.

To include this change in the SCOAP program requires a change to the way in which node C/O values are initialized. (The same effect can be achieved without any changes to the algorithm, by adding a cell which "latches" to the desired state, and by setting the depth factor of the latch elements to zero. However, an extra cell is artificially added to the circuit if this is done.) Including this small change to SCOAP can make a large difference in the C/O values which are obtained. If one treats the power supply nodes as normal inputs, which can be toggled at will, then the scores can be much lower than they would be if these nodes are correctly modelled. This is especially true for the SET and RESET lines of flip-flops.

Consider the example in Fig. 4.1. in which D-type flip/flops are used with the RESET signal fixed. If VCC and GND are treated as primary input nodes, the C/O values from SCOAP come out as in Fig. 4.1b. The VCC node, which is fixed at a logical "1", is shown to be easily controllable to a logical "0", and vice versa for the GND node. This is because these nodes are initialized by the program to be ideally controllable to "0" and "1". If a modification is made to the way the program initializes the fixed-value nodes, such that they are ideally controllable to their actual value, and uncontrollable to the other value, then the results come out as in Fig. 4.1c. (Note that this involves some minor changes to the SCOAP program itself, and that the fixed-value nodes must be identified as such in the input file which describes the circuit to the program.)

4.2 : Combinational Cell Modelling

The SCOAP C/O values are very sensitive to the models used to represent the gates in the circuit. A slightly different gate model can completely change the C/O values of the circuit. Generally, the more degrees of freedom (optional ways to

(b) VCC & GND as input nodes

NETWORK NODE NAMES	CC0	CC1	CO	SC0	SC1	SO
A	1	1	9	0	0	2
B	2	4	6	1	1	1
C	2	7	6	1	2	1
CK	1	1	12	0	0	3
D	5	7	3	2	2	0
GND	1	1	12	0	0	3
VCC	1	1	9	0	0	2
Y	2	10	0	0	2	0

(c) VCC & GND as fixed-value nodes

NETWORK NODE NAMES	CC0	CC1	CO	SC0	SC1	SO
A	1	1	14	0	0	3
B	4	4	11	1	1	2
C	7	7	8	2	2	1
CK	1	1	19	0	0	4
D	12	12	3	3	3	0
GND	1	99999	99999	0	99999	99999
VCC	99999	1	14	99999	0	3
Y	2	15	0	0	3	0

Fig. 4.1 : Power supply nodes as inputs and as fixed-value nodes

obtain a testability score) allowed in a gate model, the lower the scores will be. The more restrictive the models, the higher the scores will be. Also in general, the more complex the model, the higher the scores will be. Consider a two input Exclusive-OR gate, as in Fig. 4.2a. If a single cell is used to model it, then the C/O values for the cell alone are as shown in Fig. 4.2b. If the cell is modelled using AND, OR, and NOT gates, as in Fig. 4.2c, then the C/O values for that model are as in Fig. 4.2d. While both circuits achieve the same function, the second one appears to be more complex because of the greater number of gates and nodes.

4.3 : Sequential Cell Modelling

Consider, for example, a master/slave D-type flip/flop which requires separate inputs for both a clock signal and its inverse in order to propagate the value at the D-input to the output. If the designer always derives the inverted clock signal directly from the original clock signal, then there is no need to add the controllabilities of both clocks when calculating the controllability of the outputs. If this dependency is not taken into account, the output C/O values could be more than double what they should be.

This is illustrated by the example in Fig. 4.3a, which is a small shift register feedback circuit in which the D-type flip/flops require the clock and its inverse. The C/O values would show even greater differences between those in Fig. 4.3b and 4.3c if node CLK had not been a primary input, but had been some internal clock signal not directly available to an outside driver.

(a) $C = A \oplus B$

NETWORK NODE NAMES	CC0	CC1	CO	SC0	SC1	SO
A	1	1	2	0	0	0
B	1	1	2	0	0	0
C	3	3	0	0	0	0

(b)

(c) $C = \bar{A}B + A\bar{B}$

NETWORK NODE NAMES	CC0	CC1	CO	SC0	SC1	SO
A	1	1	6	0	0	0
ABN	2	4	3	0	0	0
AN	2	2	5	0	0	0
B	1	1	6	0	0	0
BAN	2	4	3	0	0	0
BN	2	2	5	0	0	0
C	5	5	0	0	0	0

(d)

Fig. 4.2 : Modelling of Exclusive-OR gate

(a) Example circuit

(b) Independent clocks assumption

NETWORK NODE NAMES	CC0	CC1	CO	SC0	SC1	SO
CLK	1	1	57	0	0	8
CLKB	2	2	57	0	0	8
DIN	31	2	18	4	0	3
FB	29	58	20	4	8	3
I	1	1	48	0	0	7
OUT1	37	8	12	5	1	2
OUT2	43	14	6	6	2	1
OUT3	49	20	0	7	3	0

(c) Dependent clocks assumption

NETWORK NODE NAMES	CC0	CC1	CO	SC0	SC1	SO
CLK	1	1	25	0	0	8
CLKB	2	2	25	0	0	8
DIN	15	2	6	4	0	3
FB	13	26	8	4	8	3
I	1	1	20	0	0	7
OUT1	17	4	4	5	1	2
OUT2	19	6	2	6	2	1
OUT3	21	8	0	7	3	0

Fig. 4.3 : Testability value sensitivity to gate model changes

If input dependencies are known for certain, then the gate models can be modified to produce lower scores. If such dependencies are not known (or if the design rules which govern them violated), then the more restrictive model applies, resulting generally in higher scores.

The handling of combinational and sequential gates by SCOAP breaks down for edge-triggered sequential gates. Because a transition is required rather than a logical level, the combinational C/O model must be adjusted to include them. It is not possible to apply both a logical "0" and "1" at the same time to the same node, so the C/O equations for such gates (for propagating the value of the D-input to the Q-output, for example), so "clock cycles" are used, one for each level. This super-imposing of time frames mixes the combinational and sequential testability, making the scores much harder to interpret than the case where the clock is only level sensitive.

Certain sequential gates can not be integrated into the standard SCOAP models, such as the toggle flip/flop, or counter gate. Because of the nature of the truth table, it is not possible to calculate the controllabilities of the output unless the previous state of the gate is known. The previous state data is used in the calculation of the observabilities of the inputs, as explained in Chapter 3. If one were to include the previous state information in the controllability equations, one could end up in an infinite loop: being unable to find the controllability of a node until the controllability being calculated is known. If the gates have set and clear lines, then the required previous state can be obtained using these inputs. However, it is

J-K Flip/Flop

J	K	CK	Q_{n+1}	
X	X	0	Q_n	*
0	0	1	Q_n	*
0	1	1	0	
1	0	1	1	
1	1	1	$\overline{Q_n}$	*

Truth Table

$$CC0(Q) = CC0(J) + CC1(K) + CC1(CK)$$

$$CC1(Q) = CC1(J) + CC0(K) + CC1(CK)$$

$$SC0(Q) = SC0(J) + SC1(K) + SC1(CK) + 1$$

$$SC1(Q) = SC1(J) + SC0(K) + SC1(CK) + 1$$

Controllability Equations for the J-K Flip/Flop

The states marked with an asterisk cannot be modelled in the controllability equations for Q.

Fig. 4.4 : The J-K Flip/Flop Problem

possible that in actual circuit designs these input may not be used, resulting in an "untestable" structure according to the SCOAP program.

The J-K flip/flop can be partially modelled in SCOAP, for the case where $J="0"$, $K="1"$ and for $J="1"$, $K="0"$ (see Fig. 4.4). In these states, the output is set to either "0" or "1" whenever the gate is clocked. The controllability equations do not take into account the $J="0"$, $K="0"$ or $J="1"$, $K="1"$ states because the previous state information is not available.

4.4 : Treatment of Hierarchical Circuits

In SCOAP it is necessary to expand a high level circuit description to a detailed gate level description with each node identified separately. It is possible to have a high level circuit description language which is expanded to the detail level by a translator program, but the C/O computations must be done at the gate level. The derivation of C/O equations for a block of logic can be calculated manually, by leaving the C/O relations in equation form and not assigning values. This is demonstrated by the sequential example in Fig. 4.5 (a four-bit shift register), and by the combinational example in Fig. 4.6. However, it is necessary to consider how the logic block is actually implemented. Two logic blocks which perform exactly the same logic function can produce different SCOAP scores. A more complex implementation of a particular function can be expected to have higher C/O values than a simpler one (note the difference in scores between the two different Exclusive-OR models in Fig. 4.2).

$$\begin{aligned}
 SC0(Y1) &= SC0(X) + SC0(CK) + SC1(CK) + 1 \\
 SC1(Y1) &= SC1(X) + SC0(CK) + SC1(CK) + 1 \\
 SC0(Y2) &= SC0(X) + 2(SC0(CK) + SC1(CK)) + 2 \\
 SC1(Y2) &= SC1(X) + 2(SC0(CK) + SC1(CK)) + 2 \\
 SC0(Y3) &= SC0(X) + 3(SC0(CK) + SC1(CK)) + 3 \\
 SC1(Y3) &= SC1(X) + 3(SC0(CK) + SC1(CK)) + 3 \\
 SC0(Y4) &= SC0(X) + 4(SC0(CK) + SC1(CK)) + 4 \\
 SC1(Y4) &= SC1(X) + 4(SC0(CK) + SC1(CK)) + 4
 \end{aligned}$$

$$\begin{aligned}
 SO(X) = \min(& SO(Y1) + SC0(CK) + SC1(CK) + 1, \\
 & SO(Y2) + 2(SC0(CK) + SC1(CK)) + 2, \\
 & SO(Y3) + 3(SC0(CK) + SC1(CK)) + 3, \\
 & SO(Y4) + 4(SC0(CK) + SC1(CK)) + 4)
 \end{aligned}$$

$$\begin{aligned}
 SO(CK) = \min(& SO(Y1) + SC0(Y1) + SC1(X) + SC0(CK) + SC1(CK) + 1, \\
 & SO(Y1) + SC1(Y1) + SC0(X) + SC0(CK) + SC1(CK) + 1, \\
 & SO(Y2) + SC0(Y2) + SC1(X) + 2(SC0(CK) + SC1(CK)) + 2, \\
 & SO(Y2) + SC1(Y2) + SC0(X) + 2(SC0(CK) + SC1(CK)) + 2, \\
 & SO(Y3) + SC0(Y3) + SC1(X) + 3(SC0(CK) + SC1(CK)) + 3, \\
 & SO(Y3) + SC1(Y3) + SC0(X) + 3(SC0(CK) + SC1(CK)) + 3, \\
 & SO(Y4) + SC0(Y4) + SC1(X) + 4(SC0(CK) + SC1(CK)) + 4, \\
 & SO(Y4) + SC1(Y4) + SC0(X) + 4(SC0(CK) + SC1(CK)) + 4)
 \end{aligned}$$

Note: The combinational controllability and observability equations are similar, except that the depth factor terms are all zero.

Fig. 4.5 : Four-bit shift register C/O equations

$$CC0(Y1) = \min(\begin{aligned} &CC0(X1) + CC0(X3) + 3, \\ &CC0(X1) + CC0(X4) + 3, \\ &CC0(X2) + CC0(X3) + 3, \\ &CC0(X2) + CC0(X4) + 3 \end{aligned})$$

$$CC1(Y1) = \min(\begin{aligned} &CC1(X1) + CC1(X2) + 2, \\ &CC1(X3) + CC1(X4) + 2 \end{aligned})$$

$$CC0(Y2) = \min(\begin{aligned} &CC1(X1) + CC1(X2) + 3, \\ &CC1(X3) + CC1(X4) + 3 \end{aligned})$$

$$CC1(Y2) = \min(\begin{aligned} &CC0(X1) + CC0(X3) + 4, \\ &CC0(X1) + CC0(X4) + 4, \\ &CC0(X2) + CC0(X3) + 4, \\ &CC0(X2) + CC0(X4) + 4 \end{aligned})$$

$$CO(X1) = \min(\begin{aligned} &CO(Y1) + CC1(X2) + CC0(X3) + CC0(X4) + 2, \\ &CO(Y2) + CC1(X2) + CC0(X3) + CC0(X4) + 3 \end{aligned})$$

$$CO(X2) = \min(\begin{aligned} &CO(Y1) + CC1(X1) + CC0(X3) + CC0(X4) + 2, \\ &CO(Y2) + CC1(X1) + CC0(X3) + CC0(X4) + 3 \end{aligned})$$

$$CO(X3) = \min(\begin{aligned} &CO(Y1) + CC0(X1) + CC0(X2) + CC1(X4) + 2, \\ &CO(Y2) + CC0(X1) + CC0(X2) + CC1(X4) + 3 \end{aligned})$$

$$CO(X4) = \min(\begin{aligned} &CO(Y1) + CC0(X1) + CC0(X2) + CC1(X3) + 2, \\ &CO(Y2) + CC0(X1) + CC0(X2) + CC1(X3) + 3 \end{aligned})$$

Note: The sequential controllability and observability equations are similar, except that the depth factor terms are all zero.

Fig. 4.6 : Example of combinational cell made of separate gates

4.5 Bi-Directionality

SCOAP, in its present form, handles only uni-directional signal flow circuits. Because it does not account for bi-directional gates, it excludes a large and important class of circuits and some very effective design for testability techniques. The program does include a means of analysing Wired-OR connections, for analysing open-collector TTL designs, but it is inadequate for CMOS transmission gates.

If, for example, 90% of the logic in a bi-directional circuit is in fact uni-directional logic gates (and the remaining 10% being bus driver/receivers and tri-state latches), the SCOAP algorithm is still applicable to most of the circuit. What is needed is a means of dealing with the remaining 10% of the logic, which is probably the interface between large sections of purely uni-directional logic and one or more bi-directional buses. There is no need to produce any more output than at the present: six C/O numbers per node. But a slight change is possible to include a third state, the high impedance (Z) state, as an allowed value on only the bi-directional nodes.

The "node" can be treated, in a uni-directional circuit, as an element, with only one input "branch" and one or more output branches, which passes along the value on the input to all of the outputs. The node provides the signal flow in one direction, always the same direction, from gate-to-gate. This distinction between nodes and their respective branches is simply a means of keeping track of all the signal paths to and from a given node. The controllability and observability values for the node are the same for each of its branches to the gates which it connects. This is shown in Fig. 4.7.

$CC0(outputs) = CC0(input)$
 $CC1(outputs) = CC1(input)$
 $CO(input) = \min(CO(outputs))$
 $SC0(outputs) = SC0(input)$
 $SC1(outputs) = SC0(input)$
 $SO(input) = \min(SO(outputs))$

Fig. 4.7 : Considering the Node as a Logic Element

Fig. 4.8 : A Tri-State Node as a Logic Element

Consider the uni-directional bus node, B, with tri-state drivers and receivers, as in Fig. 4.8. There are n inputs, X_1 to X_n , which are enabled by n control lines EX_1 to EX_n . There are also m outputs, Y_1 to Y_m , enabled by m control lines EY_1 to EY_m respectively.

Using the present SCOAP algorithm, one could calculate the controllabilities for X_1 to X_n , EX_1 to EX_n , and for EY_1 to EY_m , plus the observabilities for Y_1 to Y_m . It is assumed that while any number of receivers may be enabled, only one driver at a time is enabled. The junction between the drivers and receivers would normally be treated as a Wired-OR connection, as mentioned earlier (see Fig. 4.8a). This artificially adds n nodes to the circuit, because the Wired-OR connection is not actually a gate. Also, this method does not account for the need to guarantee that one and only one driver at a time is enabled.

Rather than adding the artificial Wired-OR gate, an alternative method is to instead treat node B as if it were a special logic element with $(n+m)$ ports, which can be inputs or outputs at different times. If drivers and receivers are paired, which is often the case in a bus-oriented design, then these bi-directional node logic elements are on the boundaries between the various blocks of unidirectional logic, as in Fig. 4.9.

Controlling node B in the alternative method involves not only propagating the desired value to it via the uni-directional logic, but also includes enabling one and only one driver and disabling all of the others, that is

Fig. 4.9 : Bi-Directional Nodes as Logic Block Boundaries

setting the other inputs to B to the high-impedance state. For the driver controlling node B, the controllability of B is a function of the controllability of the output of that driver, the controllability to enable that particular driver, and the controllability to disable all the other drivers connected to B. Note that some driver designs do not necessarily have an explicit "enable" node. If there is always an explicit "enable", then it is possible to simply add in the 0- and 1-controllabilities of the "enable" nodes for the drivers.

It is necessary to have three rather than two controllability figures for the output node of each driver on the "bus" node B:

CC0 - Combinational Controllability to Logical "0"
 CC1 - Combinational Controllability to Logical "1"
 CCZ - Combinational Controllability to High Impedance "Z"

If the output nodes of each driver were not connected together, we would have n separate nodes for n drivers, and 3n controllability numbers:

$CC0_i(B)$ - CC0 for the ith driver connected to B

$CC1_i(B)$ - CC1 for the ith driver connected to B

$CCZ_i(B)$ - CCZ for the ith driver connected to B

Once all of these drivers are connected together, for one particular driver to force a "0" or "1" on node B, without contention, all other drivers must be disabled. However, it is highly unlikely that a designer will intentionally design a circuit such that it is possible to enable more than one driver at a time. It is much more likely that the enable/disable controls on these drivers are interdependent.

For example, in an internal bus in a microprocessor, each driver enable may be on a unique decoded address line. Only one address is applied at a time, and therefore only

one driver can be enabled at a time. To include the sum of the controllabilities to disable all of the other drivers which are connected to the node would give a pessimistic estimate of testability. Considering only the difficulty in enabling just one driver may give too optimistic an estimate, since it may not have included controlling all of the logic necessary to disable some of the other drivers. It is more realistic to consider the degree of difficulty in enabling one particular driver, plus the maximum degree of difficulty in disabling any of the remaining drivers on the node. Deactivating the most inaccessible driver will also deactivate the easier ones in a design where the enable lines are interdependent.

The controllability of the node B will be the controllability of setting B to the desired value via the easiest route. This is the minimum of the controllabilities of B via each of the drivers connected to it, and for each case, the maximum degree of difficulty in disabling (i.e. controllability to the Z state) any of the other drivers:

$$CC0(B) = \min(CC0_i(B) + \max(CCZ_j(B))), \text{ for all } i=1,n, \text{ for all } j=1,n, \text{ } j \text{ does not equal } i \quad (4.1)$$

$$CC1(B) = \min(CC1_i(B) + \max(CCZ_j(B))), \text{ for all } i=1,n, \text{ for all } j=1,n, \text{ } j \text{ does not equal } i \quad (4.2)$$

Note that the calculation for controllability of the complete node B to the high impedance state is not necessary. We always want to have B set in a known "0" or "1" state.

It is possible to observe node B through any of the m receivers connected to it. For each of the m receivers:

$CO_k(B)$ - Combinational Observability of B via the kth receiver

The observability via the k th receiver includes the controllability necessary to enable that receiver so the signal can pass through it. For the observability of node B considering all receivers:

$$CO(B) = \min(CO(Y_k) + f_{cok}), \text{ for all } k=1,m \quad (4.3)$$

where $CO(Y_k)$ is the observability of the output of the k th receiver, and f_{cok} is the controllability required in order to sensitize the path through that particular receiver.

It may, in fact, be optimal to observe the node B through more than one receiver, if the combination somehow reduces the "distance" to a primary output. When we are considering each receiver in turn, we don't care if the others are enabled or not - unlike the drivers. If a combination is optimal, then it will be encountered by considering each one in turn.

There are some rules regarding gates which are allowed to be connected to bi-directional nodes:

1. All bi-directional devices must have at least one dedicated input-only node. It can be an "enable" or a "direction" control. Bi-directional buffers have may have both. All bi-directional devices need an output disable of some sort.
2. All bi-directional devices must have at least one bi-directional node (with a uni-directional node each for input and output), or two bi-directional nodes if the signal path is not split (as in a transmission gate). Of course, data can only flow in one direction at a time.
3. Bi-directional devices need not have dedicated output-only nodes (eg. the transmission gate) if there are at least two bi-directional nodes.
4. Only the inputs of uni-directional gates may be connected to bi-directional nodes. When calculating the observability of the node, the path through the gate must also be considered.
5. If the bi-directional node, B , is also a primary input and output, then it should be initialized as both, since it is directly accessible to be controlled or observed, ie.:

$$\begin{aligned} CC0(B) &= 1, & CC1(B) &= 1, & CO(B) &= 0 \\ SC0(B) &= 0, & SC1(B) &= 0, & SO(B) &= 0 \end{aligned}$$

Because controllabilities are still only functions of other controllabilities, and observabilities are functions of controllabilities and other observabilities, the basic SCOAP algorithm is extended, rather than changed. It is still possible to calculate all of the circuit controllabilities iteratively until they stabilize, and then separately calculate the observabilities afterwards.

Having described the method used for extending SCOAP version 2.0 to treat bi-directional nodes and gates, the implementation is now a matter of rewriting the algorithmic processor subroutines, and changing the format of the input and cell library files. This will be a fairly time consuming task - and will not be started until the next release of SCOAP is available from Berkeley. The intention of the discussion here is to establish the procedures to be used.

Two detailed bi-directional circuit examples are included in Appendix 2, for clarification of the algorithm and to illustrate how it would work in a program.

CHAPTER 5 : ANALYSIS & INTERPRETATION OF RESULTS

What is ideally desireable in interpreting the results of a testability analysis program, such as SCOAP, is a one-to-one correlation between real testing problem areas in a given circuit and testability analysis scores which point directly to those areas. For example, the sequential scores should correspond exactly to the number of sequential test steps required to access and observe a particular node. A high value indicates a very long test sequence, ie. a difficult node to test. A score of infinity represents an untestable node.

When ad-hoc design for testability techniques are being used in conjunction with the SCOAP program, the nodes which have the highest scores are the first to be considered. By applying some of the techniques described in section 2.2, and running SCOAP on the new circuit, one should be able to see an improvement in the testability scores - particularly for those nodes that the ad-hoc measures were specifically intended for. The scores for other nodes will be affected as well, due to the interdependencies of the C/O values. Often it is necessary to trace through the circuit by hand to find the source of the testing problem, since it may not lie directly at the place where the highest score was found. By making several iterations it is possible to find an optimum set (or at least a satisfactory set) of ad-hoc measures which will improve the testability. An example which shows how SCOAP can be used to improve the testability of a circuit is given in section 5.4. What follows below is a discussion of some of the problems in analysing the SCOAP output data.

5.1 : Floating Output Nodes

Input nodes are usually not allowed to float (especially in CMOS), unless there is an internal pull-up or pull-down circuit in the gate itself. Free floating input nodes in SCOAP are uncontrollable, and will cause other nodes in the circuit to be uncontrollable as well. Thus a design error will be flagged out by the SCOAP routines. Output nodes, however, do not always need to be connected in order for the circuit to function. If the sole output node of a gate is left unconnected, there is an obvious error. SCOAP will flag this out as an unobservable node.

If there is more than one output (eg. the complimentary outputs of a sequential latch), it is not necessarily an error to leave one unconnected. While it is true that the unconnected output is unobservable, the circuit can be tested by means of the other outputs. When one is examining the unobservable nodes in the SCOAP results, one has to differentiate between the nodes which are intentionally unconnected from untestable nodes which indicate serious testing problems.

5.2 : Fault Modeling

The most common fault model used in fault simulation of digital circuits is the "stuck-at" model, in which it is assumed that all of the anticipated faults which can occur inside a gate can be modelled by faults which cause inputs or outputs to be "stuck" at a fixed value. It is also assumed that if one can generate a set of test vectors which will detect all possible stuck-at faults, then that test sequence will detect the physical faults which are being modelled. The fault simulator indicates how well a particular test sequence covers the stuck-at type faults in a given circuit. It is desirable that the testability analysis program should indicate how testable the circuit is, given that stuck-at fault models are being used.

(a) Circuit example with truth table

(b) SCOAP scores for above circuit

NETWORK	NODE	NAME	CC0	CC1	CO	SCO	SC1	SO
A			3	2	8	0	0	0
B			3	4	7	0	0	0
C			5	2	5	0	0	0
D			3	6	4	0	0	0
E			4	2	8	0	0	0
X1			1	1	9	0	0	0
X2			1	1	10	0	0	0
X3			1	1	9	0	0	0
Y			10	2	0	0	0	0

(c) Equivalent circuit with minimal logic implementation

Fig. 5.1 : Circuit with redundant nodes

Consider the circuit in Fig. 5.1a, which has the SCOAP scores shown in Fig. 5.1b. If one were to attempt to generate a test for this circuit one would find that the fault "node A stuck-at-1" could not be detected with any test vector. This is because node A is in fact redundant. The same logical function can be achieved with the circuit of Fig. 5.1c. SCOAP does not explicitly indicate any testing problem associated with node A because the program, examining nodes strictly at the local level, does not account for redundant logic or reconvergent fanouts. Node A appears to SCOAP as a testable node because it can be controlled via inputs X2 and X3, and observed via output Y.

5.3 : Data Reduction

The raw testability scores are necessary in order to locate the testing problems, but there is usually too much data to get an idea of the testability of the entire circuit, or the overall effect of using design for testability measures.

Funatsu et al [23] have taken the SCOAP algorithm for calculating CC0, CC1, and CO numbers into their "calculus of testability measure". Essentially, a vector sum is made of the controllability and observability of a logic block, based on the combinational C/O values of each node in the block, and the magnitude taken as a measure of testability. They also have used a normalizing factor, based on the number of nodes in the functional block (see Fig. 5.2). Empirical scaling factors appear everywhere, indicating that the whole analysis is subject to much experimentation, using well known and new designs as test subjects. The analysis tool is "tuned" by adjusting these scaling factors, until some agreement is reached with actual experience in testing the devices. Because the correlation process is so dependent on

$$\begin{aligned}
T(FB) &= \sqrt{\{W_c \cdot C(FB)\}^2 + \{W_o \cdot O(FB)\}^2} \\
&= \sqrt{\{W_c \sqrt{\frac{\sum_{i=1}^n \{W_a \sum_{i=1}^n W_{oi} \cdot CC_0(N_i)\}^2 + \{W_b \sum_{i=1}^n W_{li} \cdot CC_1(N_i)\}^2\}} + \{W_o \sum_{i=1}^n W_{oi} \cdot CO(N_i)\}^2\}^2}
\end{aligned}$$

where: T(FB) = Testability of functional block FB
C(FB) = Controllability of FB
O(FB) = Observability of FB
Wc = Controllability weighting factor
Wo = Observability weighting factor
Wa = CC0 weighting factor for FB
Wb = CC1 weighting factor for FB
Woi = CC0 weighting factor for ith node of FB
Wli = CC1 weighting factor for ith node of FB
Woi = CO weighting factor for ith node of FB
n = number of nodes in FB
CC0(Ni) = SCOAP 0-controllability for Ni, the ith node of FB
CC1(Ni) = SCOAP 1-controllability for Ni, the ith node of FB
CO(Ni) = SCOAP observability for Ni, the ith node of FB

All empirical scaling factors initially set to unity.

Fig. 5.2: Testability at the functional block level

experimentation with individual designs, the mathematics is very simple.

No attempt is made to make use of the sequential figures (SC0, SC1, and S0), since the SCOAP modelling algorithm is considered to be inadequate for some sequential gates. The testability scores are purely combinational. Funatsu also combines the testability scores for each functional block in such a way that functional block testing problem areas can be easily identified. A functional block can be anywhere from a single gate to a complete IC or board in a system. But the intention of the measure is to isolate testing problems down to specific logic blocks. Some threshold testability value is chosen, and any functional block whose testability value exceeds this is subject to close scrutiny.

Mean testability values, and standard deviation values of the testability distribution are not very useful since the typical distribution for testability is not at all Gaussian. It is useful to show the spread of testability values in a histogram, where each cell represents a different level of testing difficulty, if suitable boundary values for each cell can be determined.

In the circuit example in Fig. 5.3, the vector sum approach of Funatsu et al was simplified to analyse one node at a time in the circuit (the method is illustrated in Fig. 5.4). The circuit has been found to be a difficult one to test [12]. It has a large number of internal feedback loops, reconvergent fanouts, and long series of latches which are difficult to set or reset. The testability scores were normalized by dividing by the number of nodes in the

Fig. 5.3 : Example of a difficult circuit to test [12]

(a) Vector sum of controllabilities and observabilities of node N

$$T_n(N) = \frac{\sqrt{(a_1 CO(N))^2 + (a_2 CCO(N))^2 + (a_3 CCl(N))^2}}{a_4^n}$$

(b) Equation for normalized testability $T_n(N)$ for node N

where : a_1 through a_4 are empirical scaling factors, set to unity, and n is the number of nodes in the circuit.

Fig. 5.4 : Normalized testability at the node level

circuit. Ten testing categories from "trivial" to "untestable" were chosen. The boundaries chosen were in powers of two, so that the cells are exponentially related. The histogram, and a small amount of statistical data are presented in Fig. 5.5. In the complete output listing there is also a list of all the nodes in the circuit, sorted by the normalized testability, and a list of all nodes in the "untestable" category, which allows the designer/test engineer to examine specific nodes.

A normalizing factor was chosen to give an indication of the testability of each node relative to the size of the circuit. In the example, the scaling factor is the total number of nodes in the circuit. Because one expects to find higher scores for larger circuits, it would be desirable to have a means of comparing circuits of different sizes for their relative testability. The normalizing factor could take into account the number of primary inputs and outputs, or the number of gates, if desired.

Kovijanic's approach [12] is to derive a single value for an Inherent Testability Figure of Merit (ITFOM), that is to reduce all of the data from a TESTSCREEN run to one number. He claims the number can be correlated to the amount of effort required by either manual or automated test pattern generation for the circuit being considered. This is a useful result, but the designer still wants to know exactly where the problem areas lie. And the testing engineer still wants to know exactly how many test sequence steps are required to stimulate and observe the most inaccessible portions of the circuit. However, the single

figure value can be used as a goal or criterion for design, and as an indication of when the testability has reached an acceptable level and one can then begin to design the test program itself.

5.4: Analysis Example

To demonstrate how the SCOAP testability scores are analysed for a reasonably large circuit, consider once again the example in Fig. 5.3. After running the circuit through SCOAP, and applying the vector sum testability measure described in Fig. 5.4, the distribution of the nodal testabilities is as shown in Fig. 5.5

In the example, most of the "untestable" nodes are unused outputs of latches, in which both Q and its inverse are provided, but only one is needed. However, a few nodes in this category are centred around output OP14, which latches itself into the "0" state because of the way it is connected. The nodes affected are indicated by the thick lines in Fig. 5.3. The connection of the latch feeding OP14 is untestable, because once the D-flip/flop clears itself, it cannot be set again unless the whole circuit is shut down. Also, there is no way of knowing which state the latch will be in when power is resupplied to the circuit.

There is also a group of untestable nodes around OP12, the NAND gate, the D-type flip/flop, and the Exclusive-OR gate which drive it. This portion of the circuit is untestable because the Q-output of the D-type flip/flop is fed back to its D-input through the Exclusive-OR gate. To calculate the controllability of its output, the controllabilities of both of the inputs must be known. The program gets caught in a loop where it must know the controllability of the Q-output of the flip/flop in order to calculate the controllability of that same Q-output. In testing, it would not be possible to predict what the state of OP12 would be on power-up.

The highest finite scores are clustered around OP10, and the five D-type flip/flops from which it is derived. Upon inspection of the C/O scores for the flip/flop outputs, it is noted that either the controllability to "1" or "0" is very high, but not both. (To see this, one would have to refer to the raw C/O data produced by SCOAP, since this information is lost when the histogram data is produced.) The controllability to "0" is difficult where the PRESET input is connected and the CLEAR input is not used, and the controllability to "1" is high in the opposite case.

To improve the testability of this circuit, three changes were made which would not affect the normal operation of this circuit: a) OP14 was isolated from the flip/flop CLEAR input by means of an OR gate. The other input was connected to a test point, TP21; b) the previously unused CLEAR input to the flip/flop driving OP12 was connected to another testpoint, TP22; and c) the same testpoint, TP22, was connected to the unconnected CLEAR/PRESET lines of the five D-type flip/flops which determine the value of OP10. The changes to the circuit are illustrated in Fig. 5.6, and the new testability distribution is shown in Fig. 5.7.

The only remaining untestable nodes are the unused extra outputs from flip/flops. The testability of the most inaccessible nodes has been greatly improved, but the bulk of the nodes still fall in the HARD to MEDIUM-HARD category. It would easily be possible to iterate with more changes and SCOAP runs to further improve the testability of this circuit.

Figures 5.3, 5.5, 5.6, and 5.7 illustrate that the SCOAP results for a large circuit, when used in conjunction with the schematic diagram, yield some useful information about the testability. The highest C/O values indicate potential testing problem areas, which the circuit designer then examines and decides what action (if any) should be taken.

Fig. 5.6 : Example circuit with changes to improve testability

NORMALIZED TESTABILITY		NODES	PERCENT	HISTOGRAM
PERCENTILE	TESTABILITY			
90%ILE	918.8613	14	12.07 %	*****
75%ILE	560.1304	0	0.00 %	
50%ILE	263.0684	7	6.03 %	*****
25%ILE	201.9926	22	18.97 %	*****
10%ILE	29.4279	37	31.90 %	*****
AVERAGE	382.0103	13	11.21 %	*****
		0	0.00 %	*****
		12	10.34 %	*****
		7	6.03 %	*****
		4	3.45 %	***

NORMALIZED TESTABILITY		NODES	PERCENT	HISTOGRAM
PERCENTILE	TESTABILITY			
90%ILE	918.8613	14	12.07 %	*****
75%ILE	560.1304	0	0.00 %	
50%ILE	263.0684	7	6.03 %	*****
25%ILE	201.9926	22	18.97 %	*****
10%ILE	29.4279	37	31.90 %	*****
AVERAGE	382.0103	13	11.21 %	*****
		0	0.00 %	*****
		12	10.34 %	*****
		7	6.03 %	*****
		4	3.45 %	***

TOTAL NODES:	116
INPUTS:	9
OUTPUTS:	13

Fig. 5.7 : Testability distribution after testpoints added

CHAPTER 6 : SUMMARY, CONCLUSIONS & RECOMMENDATIONS

The problems of testing digital circuits have been discussed, as well as various approaches to improving testability, with emphasis on the use of an automated testability analysis program. The bulk of the work described herein was conducted using the SCOAP program, because of its availability.

6.1 : Summary

The SCOAP testability analysis program was first released by the University of California (Berkeley) in early 1981. The program was installed and debugged by the author on a VAX 11/780 at Mitel Corporation in Kanata. The cell library provided with the program was extensively modified to accomodate the Uncommitted Logic Array gate elements used by Mitel. A translator program was written so that circuits which had been entered into a network description language file could be converted to a SCOAP compatible input file. Another program was written which would reduce the data from the SCOAP output file in order to make it easier to interpret the results. Some improvements to the SCOAP algorithm, listed below, have been proposed and tested (The implementation of the actual code itself is being co-ordinated by the Department of Electrical Engineering and Computer Science, University of California at Berkeley in order to have a common version of the program available for distribution.):

1. Nodes which have fixed logical values must be initialized as such (ie. perfectly controllable to one value, and totally uncontrollable to the other). Otherwise they will be initialized as primary inputs (which can be toggled between logical "0" and "1" at will), resulting in much lower C/O values in the circuit, and possibly showing untestable nodes as testable.

2. Unused output nodes which are intentionally left not connected should be deleted from the output listings since they do not present any testing problem.
3. Bi-directionality is a desirable feature when one is working with CMOS gates. The method described in Chapter 4, which introduces a high impedance state to the C/O equations for the bi-directional nodes, and adds in the cost of disabling non-talking drivers, is only one of several possible solutions to adapting the SCOAP program.

6.2 : Interpretation of SCOAP

A number of theoretical and practical examples were run in order to find ways of analyzing the C/O values generated by SCOAP. Since the program was heuristic, the meaning of the scores obtained was not clear. In many cases the results produced by SCOAP did not correspond to the actual testability. The findings are summarized below:

1. The SCOAP program does not predict the length of the test sequence required for any given circuit, as would be most desirable for a testability analysis program.
2. Because the SCOAP program calculates testability scores at the local node level, it does not account for redundancy or reconvergent fanout in the circuit. Nodes for which stuck-at type faults would not be detectable, because they are masked by redundancy, will not be flagged by SCOAP. In order to detect such redundancy it is necessary to know the interdependencies of all the nodes in the circuit, as is the case with circuit simulation and automatic test generation.
3. The SCOAP program is able to detect unconnected inputs and outputs of cells in a circuit, and identify portions of the circuit which cannot be initialized. This can be used to perform a fast check for errors in the circuit (although a logic simulator will detect the same kinds of mistakes).
4. The SCOAP program is very sensitive to errors in the gate models, and all gate or cell library entries should be thoroughly tested before being used in actual circuit runs.

5. When analyzing the SCOAP C/O scores, one first considers the uncontrollable and unobservable nodes, (marking these scores directly on a schematic diagram of the circuit is very helpful) and tries to find what causes these nodes to be labeled as untestable by the program. Next, one examines the nodes which have finite, but very high, C/O values, and so on. It is very difficult to attach much meaning to the lowest scores, except that in general, they represent nodes which are not significant testing problems.
6. The raw testability data is necessary in order to know which nodes of the circuit would benefit most from the addition of testpoints, additional test logic, or whatever. However, it is useful to have an overall testability objective, for example an indication of the cost of testing the whole circuit. This indicator would be used to determine when the testability improvement effort should be ceased, and test generation effort started. Because of the highly empirical nature of testing, a mathematically simple means of reducing the data, and a large number of experimental circuits on which to run and adjust the program are needed.

6.3 : Conclusions

SCOAP cannot be used to predict the length of the test sequence for a circuit, and it does not detect all of the possible types of untestable circuits. However, the SCOAP C/O scores are still useful as a CAD tool when used in conjunction with ad-hoc testability improvement techniques. In general, high scores indicate testing problem areas. When one is attempting to improve the testability of a circuit using ad-hoc measures, the nodes with the highest C/O values are the first to be considered. The improvement in testability is judged by the changes in the C/O values which resulted. Improving the testability of a highly untestable node may also improve the testability of the surrounding circuitry as well.

6.4 : Recommendations For Future Work

1. It is desirable to adapt the SCOAP program to handle hierarchical circuits. It is possible to manually calculate the C/O equations for large functional blocks which are composed of smaller ones with known C/O relations. The present program works strictly at the local node level, and does not save any information on the function of the circuit which it is analysing. Since the program initially knows the topology of the circuit, and the function of each of its components, it is possible to have it derive the C/O relationships based on this knowledge - as may be done with the TMEAS and CAMELOT programs. The library data format (and all affected subroutines) would need to be modified extensively. However, building in the capability to handle hierarchical circuits will not improve the program's basic inability to deal with redundancy.
2. The SCOAP program, in calculating the C/O values for each node, processes enough information in calculating the testability values for each node to perhaps derive the actual test vectors themselves. The C/O values are based on the assumption of the most easily performed test. The final values indicate that there is some set of test vectors which satisfy those numbers. Note that SCOAP itself was originally intended as a pre-processor for a fault simulator or automated test pattern generator, and intentionally runs much faster (and stores much less information) than either of these. But the algorithms used to derive minimum C/O values may also yield minimum test vector patterns if extended.

APPENDIX 1 : SCOAP EXAMPLE

To clarify the use of SCOAP a small but detailed example is provided here. Suppose we wish to perform a testability analysis on the circuit in Fig. A1.1 (borrowed from [10]), which contains three sequential gates and two combinational gates.

First of all, the C/O equations will have previously been determined for each different type of gate in the circuit. (The equations for a two-input OR gate and a positive edge triggered D-type flip/flop were shown in Figs. 3.1 and 3.2.) Using the C/O matrix definitions, the gate equations will have been translated into the standard cell format, as shown in Fig. A1.2 for a two-input Exclusive-OR gate.

The user encodes his circuit into an input file which conforms to the format for SCOAP. The user indicates which nodes are primary inputs and primary outputs. Then the gate data is entered, one line at a time, first giving the cell type number, then the output and input nodes in a pre-determined order. The example circuit input file is shown in Fig. A1.3a.

Now the user is ready to run the SCOAP routines. The first program, PRESCO, which does the pre-processing, requests the names of the main input and output files, or asks if the user wishes to use the same ones as the last time. PRESCO then opens and builds the temporary files which contains cell data, node name to node number translation data, intermediate C/O values, etc. It also checks for syntax errors in the input file.

ALSCO, the algorithmic processor, does all of the calculations, and iterates until the C/O values have stabilized throughout the network. (Since minimum values are always used in any calculation, the solution never oscillates, but it may require many iterations to "settle".) The final C/O values are stored in one of the temporary files.

POSCO, the post-processor, is a collection of sorting routines. It takes the raw C/O data and sorts it alphabetically by node name into the main output file. It also creates six temporary files, one for each of the C/O functions, sorted by numerical value: highest scores at the beginning, lowest scores at the end. Fig. A1.3b shows the input file and the main output file for our example.

From this the user interprets the testability of each node in his circuit. For our example, the most difficult nodes to control are OUT3, the primary output node, which is at the end of the flip-flop chain, and FB, which is the farthest node from the inputs, and whose controllability depends on OUT1 and OUT3, not directly on the input value. The most difficult nodes to observe are the two primary inputs I and CLK, whose values must propagate through the entire circuit to reach the only primary output node.

This simple example does not point out the main problem when using SCOAP: interpretation. It is relatively easy to see where all of the scores came from (see the equations in Fig. A1.4), and to correlate the higher scores to the more inaccessible nodes. This example was chiefly intended to illustrate the use of the program and its cell library.

Fig. A1.1 : Example circuit for SCOAP run

Exclusive-OR Gate (C2310) : $C = A \oplus B$

$$CCO(C) = \min(CCO(A) + CCO(B) + 1, CCl(A) + CCl(B) + 1)$$

$$CCl(C) = \min(CCO(A) + CCl(B) + 1, CCl(A) + CCO(B) + 1)$$

$$CO(A) = \min(CO(C) + CCO(B) + 1, CO(C) + CCl(B) + 1)$$

$$CO(B) = \min(CO(C) + CCO(A) + 1, CO(C) + CCl(A) + 1)$$

$$SCO(C) = \min(SCO(A) + SCO(B) + 1, SCl(A) + SCl(B))$$

$$SCl(C) = \min(SCO(A) + SCl(B) + 1, SCl(A) + SCO(B))$$

$$SO(A) = \min(SO(C) + SCO(B), SO(C) + SCl(C))$$

$$SO(B) = \min(SO(C) + SCO(A), SO(C) + SCl(A))$$

	CC(A)		CC(B)		DEPTH	
	0	1	0	1	+1	OCTAL:
CC0(C)	(	1	0	1	0 1)	25
	(	0	1	0	1 1)	13
CC1(C)	(	1	0	0	1 1)	23
	(	0	1	1	0 1)	15

	CO(C)	CC(C)	CC(A)	CC(B)	DEPTH	
		0 1	0 1	0 1	+1	OCTAL:
CO(A)	(1	0 0	0 0	1 0	1)	205
	(1	0 0	0 0	0 1	1)	203
CO(B)	(1	0 0	1 0	0 0	1)	221
	(1	0 0	0 1	0 0	1)	211

SCO(C), SCl(C), SO(A), and SO(B) terms the same, except that the depth "+1" factor is zero because this is a combinational gate.

Library cell entry for the above gate:

```
*EXCLUSIVE-OR GATE, TWO INPUTS
NAME C2310 *IOLIST: OUT=1 IN=2
CC 25 M 13 23 M 15
CO 205 M 203 221 M 211
SC 24 M 12 22 M 14
SO 204 M 202 220 M 210
```

Fig. A1.2 : SCOAP C/O relations and library cell entry

```

*   THREE-STAGE FEEDBACK SHIFT REGISTER CIRCUIT
*   SMALL CIRCUIT EXAMPLE                25MAY1981
INPUT I CLK
OUTPUT OUT3
C1720 DIN I FB
C2310 FB OUT1 OUT3
C1470 OUT1 DIN CLK
C1470 OUT2 OUT1 CLK
C1470 OUT3 OUT2 CLK
END

```

a) Example input file created by the user

```

***** PROGRAM SCOAP *****
VERSION      2 0      1/1/81
*****

```

```

INPUT NETWORK READ FROM FILE "TESTX INP"
INPUT NETWORK NODE COUNT      7

```

```

CC0 = 0 COMBINATIONAL CONTROLLABILITY
CC1 = 1 COMBINATIONAL CONTROLLABILITY
CO  =  COMBINATIONAL OBSERVABILITY
SCO = 0 SEQUENTIAL CONTROLLABILITY
SC1 = 1 SEQUENTIAL CONTROLLABILITY
SD  =  SEQUENTIAL OBSERVABILITY

```

NETWORK NODE NAMES	CC0	CC1	CO	SCO	SC1	SD
CLK	1	1	25	0	0	8
DIN	15	2	6	4	0	3
FB	13	26	8	4	8	3
I	1	1	20	0	0	7
OUT1	17	4	4	5	1	2
OUT2	19	6	2	6	2	1
OUT3	21	8	0	7	3	0

b) SCOAP main output file

Fig. A1.3 : Main Input/Output files for the SCOAP example

INIT: $CC0(I)=1$ $SC0(I)=0$ All other values initialized to:
 $CC1(I)=1$ $SC1(I)=0$ $CC = CO = SC = SO = 99999$
 $CC0(CLK)=1$ $SC0(CLK)=0$
 $CC1(CLK)=1$ $SC1(CLK)=0$
 $CO(OUT3)=0$ $SO(OUT3)=0$

CC: $CC0(DIN)=CC0(I)+CC0(FB)+1$
 $CC1(DIN)=\min(CC1(I)+1, CC1(FB)+1)$
 $CC0(FB)=\min(CC0(OUT1)+CC0(OUT3)+1, CC1(OUT1)+CC1(OUT3)+1)$
 $CC1(FB)=\min(CC0(OUT1)+CC1(OUT3)+1, CC1(OUT1)+CC0(OUT3)+1)$
 $CC0(OUT1)=CC0(DIN)+CC0(CLK)+CC1(CLK)$
 $CC1(OUT1)=CC1(DIN)+CC0(CLK)+CC1(CLK)$
 $CC0(OUT2)=CC0(OUT1)+CC0(CLK)+CC1(CLK)$
 $CC1(OUT2)=CC1(OUT1)+CC0(CLK)+CC1(CLK)$
 $CC0(OUT3)=CC0(OUT2)+CC0(CLK)+CC1(CLK)$
 $CC1(OUT3)=CC1(OUT2)+CC0(CLK)+CC1(CLK)$
CO: $CO(OUT2)=CO(OUT3)+CC0(CLK)+CC1(CLK)$
 $CO(OUT1)=\min(CO(OUT2)+CC0(CLK)+CC1(CLK),$
 $\quad CO(FB)+CC0(OUT3)+1, CO(FB)+CC1(OUT3)+1)$
 $CO(DIN)=CO(OUT1)+CC0(CLK)+CC1(CLK)$
 $CO(FB)=CO(DIN)+CC0(I)+1$
 $CO(I)=CO(DIN)+CC0(FB)+1$
 $CO(CLK)=\min(CO(OUT3)+CC1(OUT3)+CC0(OUT2)+CC0(CLK)+CC1(CLK),$
 $\quad CO(OUT3)+CC0(OUT3)+CC1(OUT2)+CC0(CLK)+CC1(CLK),$
 $\quad CO(OUT2)+CC1(OUT2)+CC0(OUT1)+CC0(CLK)+CC1(CLK),$
 $\quad CO(OUT2)+CC0(OUT2)+CC1(OUT1)+CC0(CLK)+CC1(CLK),$
 $\quad CO(OUT1)+CC1(OUT1)+CC0(DIN)+CC0(CLK)+CC1(CLK),$
 $\quad CO(OUT1)+CC0(OUT1)+CC1(DIN)+CC0(CLK)+CC1(CLK))$

The same equations apply for SC's and SO's except:

1. Substitute "SC" and "SO" for "CC" and "CO" respectively.
2. Remove the "+1" depth factor from any expression above that has one.
3. Add a "+1" depth factor for any expression that does not have one.

Fig. A1.4 : Controllability/Observability equations for the example

APPENDIX 2 : BI-DIRECTIONAL EXAMPLES

(A) Bi-Directional Node Imbedded in the Circuit

Consider Fig. A2.1, which shows a bi-directional node buried inside some circuit. This could be one bit of an internal address or data bus in a microprocessor which is not directly accessible from the pins of the IC. One would expect that the testability of all of the lines of the bus would be nearly equal since they are parallel copies of the same circuit with only slight logic differences between them.

Connected to the bi-directional node, B, are three tri-state drivers: U1, U3, and U6; three tri-state receivers: U2, U4, and U5; and an input to NOR gate U7. If all of these gates are fairly deep into the circuit, none of the nodes shown are primary inputs or outputs, and the C/O values would be initialized to infinity.

In this example some of the C/O values are independent of node B. This is possible if node B is on the boundary between otherwise separate logic blocks, and B is the means of transmitting data between them. These are for the inputs and outputs to the circuit shown, and their controllabilities or observabilities can be calculated separately from the unknown quantities. For this example they are assigned arbitrary values

(those shown in Fig. A2.1):

CC0(EN1)	=	23,	CC1(EN1)	=	16	
CC0(EN2)	=	17,	CC1(EN2)	=	24	
CC0(EN3)	=	30,	CC1(EN3)	=	37	
CC0(EN4)	=	25,	CC1(EN4)	=	31,	CO(DATA2) = 30
CC0(EN5)	=	55,	CC1(EN5)	=	90,	CO(DATA4) = 25
CC0(EN6)	=	87,	CC1(EN6)	=	41,	CO(DATA5) = 47
CC0(DATA1)	=	60,	CC1(DATA1)	=	42,	CO(DATA7) = 13
CC0(DATA3)	=	99,	CC1(DATA3)	=	31	
CC0(DATA6)	=	39,	CC1(DATA6)	=	132	
CC0(TEST)	=	46,	CC1(TEST)	=	31

Fig. A2.1 : Bi-directional node in a circuit - initial C/O values

To calculate $CC0(B)$, we apply our bi-directional equations:

$$CCZ_{U1}(B) = CC0(EN1) = 23$$

$$CCZ_{U3}(B) = CC0(EN3) = 30 \quad \text{to get state "Z" from each driver to node B}$$

$$CCZ_{U6}(B) = CC0(EN6) = 87$$

$$CC0_{U1}(B) = CC0(DATA1) + CC1(EN1) + D_{U1} = 60 + 16 + 1 = 77$$

$$CC0_{U3}(B) = CC0(DATA3) + CC1(EN3) + D_{U3} = 99 + 37 + 1 = 137$$

$$CC0_{U6}(B) = CC0(DATA6) + CC1(EN6) + D_{U6} = 39 + 41 + 1 = 81$$

Therefore, to calculate the 0-controllability of node B:

$$\begin{aligned} CC0(B) &= \min(CC0_{U1}(B) + \max(CCZ_{U3}(B), CCZ_{U6}(B)), \\ &\quad CC0_{U3}(B) + \max(CCZ_{U1}(B), CCZ_{U6}(B)), \\ &\quad CC0_{U6}(B) + \max(CCZ_{U1}(B), CCZ_{U3}(B))) \\ &= \min(77 + \max(30, 87), 137 + \max(23, 87), 81 + \max(23, 30)) \\ &= \min(164, 224, 111) \\ &= 111 \end{aligned}$$

Therefore B is most easily controlled to "0" via driver U6. To calculate the 1-controllability of node B:

$$CC1_{U1}(B) = CC1(DATA1) + CC1(EN1) + D_{U1} = 42 + 16 + 1 = 59$$

$$CC1_{U3}(B) = CC1(DATA3) + CC1(EN3) + D_{U3} = 31 + 37 + 1 = 69$$

$$CC1_{U6}(B) = CC1(DATA6) + CC1(EN3) + D_{U6} = 132 + 41 + 1 = 174$$

$$\begin{aligned} CC1(B) &= \min(CC1_{U1}(B) + \max(CCZ_{U3}(B), CCZ_{U6}(B)), \\ &\quad CC1_{U3}(B) + \max(CCZ_{U1}(B), CCZ_{U6}(B)), \\ &\quad CC1_{U6}(B) + \max(CCZ_{U1}(B), CCZ_{U3}(B))) \\ &= \min(59 + \max(30, 87), 69 + \max(23, 87), 174 + \max(23, 30)) \\ &= \min(146, 156, 204) \\ &= 146 \end{aligned}$$

B is most easily controlled to "1" via driver U1. We can now calculate the controllability of DATA7 through the NOR gate U7:

$$\begin{aligned} \text{CC0}(\text{DATA7}) &= \min(\text{CC1}(\text{B}) + D_{\text{U7}} , \text{CC1}(\text{TEST}) + D_{\text{U7}}) \\ &= \min(146 + 1, 51 + 1) \\ &= \min(147, 52) \\ &= 52 \end{aligned}$$

$$\begin{aligned} \text{CC1}(\text{DATA7}) &= \text{CC0}(\text{B}) + \text{CC0}(\text{TEST}) + D_{\text{U7}} \\ &= 111 + 46 + 1 \\ &= 158 \end{aligned}$$

Controllabilities for DATA2, DATA4, and DATA5 can be obtained:

$$\text{CC0}(\text{DATA2}) = \text{CC0}(\text{B}) + \text{CC1}(\text{EN2}) + D_{\text{U2}} = 111 + 24 + 1 = 136$$

$$\text{CC1}(\text{DATA2}) = \text{CC1}(\text{B}) + \text{CC1}(\text{EN2}) + D_{\text{U2}} = 146 + 24 + 1 = 171$$

$$\text{CC0}(\text{DATA4}) = \text{CC0}(\text{B}) + \text{CC1}(\text{EN4}) + D_{\text{U4}} = 111 + 31 + 1 = 143$$

$$\text{CC1}(\text{DATA4}) = \text{CC1}(\text{B}) + \text{CC1}(\text{EN4}) + D_{\text{U4}} = 146 + 31 + 1 = 178$$

$$\text{CC0}(\text{DATA5}) = \text{CC0}(\text{B}) + \text{CC1}(\text{EN5}) + D_{\text{U5}} = 111 + 90 + 1 = 202$$

$$\text{CC1}(\text{DATA5}) = \text{CC1}(\text{B}) + \text{CC1}(\text{EN5}) + D_{\text{U5}} = 146 + 90 + 1 = 237$$

The observability of B can be calculated now:

$$\text{CO}_{\text{U2}}(\text{B}) = \text{CO}(\text{DATA2}) + \text{CC1}(\text{EN2}) + D_{\text{U2}} = 30 + 24 + 1 = 55$$

$$\text{CO}_{\text{U4}}(\text{B}) = \text{CO}(\text{DATA4}) + \text{CC1}(\text{EN4}) + D_{\text{U4}} = 25 + 31 + 1 = 57$$

$$\text{CO}_{\text{U5}}(\text{B}) = \text{CO}(\text{DATA5}) + \text{CC1}(\text{EN5}) + D_{\text{U5}} = 47 + 90 + 1 = 138$$

$$\text{CO}_{\text{U7}}(\text{B}) = \text{CO}(\text{DATA7}) + \text{CC0}(\text{TEST}) + D_{\text{U7}} = 13 + 51 + 1 = 65$$

$$\begin{aligned} \text{CO}(\text{B}) &= \min(\text{CO}_{\text{U2}}(\text{B}), \text{CO}_{\text{U4}}(\text{B}), \text{CO}_{\text{U5}}(\text{B}), \text{CO}_{\text{U7}}(\text{B})) \\ &= \min(55, 57, 138, 65) \\ &= 55 \end{aligned}$$

B is easiest to observe via receiver U2. The observability of the TEST node is:

$$\begin{aligned}\text{CO}(\text{TEST}) &= \text{CO}(\text{DATA7}) + \text{CC0}(\text{B}) + \text{D}_{\text{U7}} \\ &= 13 + 111 + 1 \\ &= 126\end{aligned}$$

The observabilities of DATA1, DATA3, and DATA6 can now be calculated:

$$\text{CO}(\text{DATA1}) = \text{CO}(\text{B}) + \text{CC1}(\text{EN1}) + \text{D}_{\text{U1}} = 55 + 16 + 1 = 72$$

$$\text{CO}(\text{DATA3}) = \text{CO}(\text{B}) + \text{CC1}(\text{EN3}) + \text{D}_{\text{U3}} = 55 + 30 + 1 = 86$$

$$\text{CO}(\text{DATA6}) = \text{CO}(\text{B}) + \text{CC1}(\text{EN6}) + \text{D}_{\text{U6}} = 55 + 41 + 1 = 97$$

As in example (A), to observe the enable lines, it is necessary to set the driver/receiver inputs to one value, and then toggle to the opposite value. If a change is detected at the output, the enable was "on", otherwise it was "off".

$$\begin{aligned}\text{CO}(\text{EN1}) &= \text{CO}(\text{B}) + \text{CC0}(\text{DATA1}) + \text{CC1}(\text{DATA1}) + \text{D}_{\text{U1}} \\ &= 55 + 60 + 42 + 1 \\ &= 158\end{aligned}$$

$$\begin{aligned}\text{CO}(\text{EN2}) &= \text{CO}(\text{DATA2}) + \text{CC0}(\text{B}) + \text{CC1}(\text{B}) + \text{D}_{\text{U2}} \\ &= 30 + 111 + 146 + 1 \\ &= 288\end{aligned}$$

$$\begin{aligned}\text{CO}(\text{EN3}) &= \text{CO}(\text{B}) + \text{CC0}(\text{DATA3}) + \text{CC1}(\text{DATA3}) + \text{D}_{\text{U3}} \\ &= 55 + 99 + 31 + 1 \\ &= 186\end{aligned}$$

$$\begin{aligned}\text{CO}(\text{EN4}) &= \text{CO}(\text{DATA4}) + \text{CC0}(\text{B}) + \text{CC1}(\text{B}) + \text{D}_{\text{U4}} \\ &= 25 + 111 + 146 + 1 \\ &= 283\end{aligned}$$

$$\begin{aligned}\text{CO}(\text{EN5}) &= \text{CO}(\text{DATA5}) + \text{CC0}(\text{B}) + \text{CC1}(\text{B}) + \text{D}_{\text{U5}} \\ &= 47 + 111 + 146 + 1 \\ &= 305\end{aligned}$$

$$\begin{aligned}\text{CO}(\text{EN6}) &= \text{CO}(\text{B}) + \text{CC0}(\text{DATA6}) + \text{CC1}(\text{DATA6}) + \text{D}_{\text{U6}} \\ &= 55 + 39 + 132 + 1 \\ &= 227\end{aligned}$$

Fig. A2.2 : Bi-directional node in a circuit - final C/O values

The final C/O values are shown in Fig. A2.2.

The 0-controllability figure of B was achieved via driver U6. The 0-controllability of U6's input, DATA6, was the lowest of the possible choices ($CC0(DATA1)=60$, $CC0(DATA3)=99$, $CC0(DATA6)=39$).

The 1-controllability of B was achieved via driver U1, which does not have the lowest 1-controllability on its input, DATA1, of all the available choices ($CC1(DATA1)=42$, $CC1(DATA3)=31$, $CC1(DATA6)=132$). But note the driver U1 is the easiest one to enable ($CC1(EN1)=16$, $CC1(EN3)=37$, $CC1(EN6)=41$).

The observability of B through receiver U2 is the best not because the output of U2, DATA2, is the most observable choice ($CO(DATA2)=30$, $CO(DATA4)=25$, $CO(DATA5)=47$, $CO(DATA7)=13$), but because the U2 path is the easiest one to sensitize.

U7 may have been intended to provide an easy means of reading the value of B - but we have found that the value of B was more easily accessed via U2. It was included in this example to show that normal uni-directional gate inputs can be connected to the node B, even though it is a bi-directional node.

(B) Bi-Directional I/O Pin

Fig. A2.3a shows a portion of an integrated circuit around a bi-directional node, PIN, which happens to be directly connected to a bonding pad of the device. Normally, this circuit would require two runs of the SCOAP programs. In the first PIN would be identified as a primary input pin, and its 0- and 1-controllabilities initialized to unity. This run would give reasonable results for the controllabilities of nodes for which PIN is an input, or

(a) Initial C/O values

(b) Final C/O values

(CC0, CC1, CO)

Fig. A2.3 : Bi-directional primary I/O node

part of the path sensitization. A second run is necessary, with PIN initialized as a primary output (observability of PIN set to zero), to obtain the observabilities of the nodes for which PIN is used as an output. The results of the two runs can be combined, taking the minimum C/O values from each into the final output.

It would be preferable to make just one run of the program, modelling nodes such as PIN as true bi-directional nodes. The proposed changes to the SCOAP algorithm will accomplish this for bi-directional nodes which are I/O pins, and those which are imbedded deep inside the circuit. Two examples are given to clarify how these changes will work.

Returning to Fig. A2.3a, some of the initial C/O values are not dependent on the unknown values for this circuit. These are the controllabilities for the inputs to the portion of logic shown, and the observabilities of the outputs which lead to other parts of the integrated circuit. Also the C/O values for PIN are initialized to the minimum possible values (since node PIN is perfectly testable). For brevity, only the combinational C/O values are shown. Sequential C/O values are calculated in the same way. For this example arbitrary values are chosen for the independent C/O numbers.

CC0(PIN) = 1	CC1(PIN) = 1	CO(PIN) = 0
CC0(E1) = 23	CC1(E1) = 19	
CC0(E2) = 17	CC1(E2) = 25	
CC0(E3) = 26	CC1(E3) = 29	
CC0(E4) = 31	CC1(E4) = 21	
CC0(E5) = 11	CC1(E5) = 24	
CC0(E6) = 28	CC1(E6) = 39	
CC0(D1) = 10	CC1(D1) = 9	
CC0(D3) = 22	CC1(D3) = 13	
CC0(D5) = 8	CC1(D5) = 15	
CO(D2) = 33	CO(D4) = 18	CO(D6) = 27

The controllabilities of the receiver outputs, D2 and D4, and the NAND gate output, D6, can be calculated:

$$CC0(D2) = CC0(PIN) + CC1(E2) + D_{U2} = 1 + 25 + 1 = 27$$

$$CC1(D2) = CC1(PIN) + CC1(E2) + D_{U2} = 1 + 25 + 1 = 27$$

$$CC0(D4) = CC0(PIN) + CC1(E4) + D_{U4} = 1 + 21 + 1 = 23$$

$$CC1(D4) = CC1(PIN) + CC1(E4) + D_{U4} = 1 + 21 + 1 = 23$$

$$CC0(D6) = CC1(PIN) + CC1(E6) + D_{U6} = 1 + 39 + 1 = 41$$

$$CC1(D6) = \min(CC0(PIN) + D_{U2}, CC0(E6) + D_{U2}) \\ = \min(1 + 1, 28 + 1) = \min(2, 29) = 2$$

The observabilities of the driver data inputs, D1, D3, and D5:

$$CO(D1) = CO(PIN) + CC1(E1) + D_{U1} = 0 + 19 + 1 = 20$$

$$CO(D3) = CO(PIN) + CC1(E3) + D_{U3} = 0 + 29 + 1 = 30$$

$$CO(D5) = CO(PIN) + CC1(E5) + D_{U5} = 0 + 24 + 1 = 25$$

The observability of the E6 input to the NAND gate:

$$CO(E6) = CO(D6) + CC1(PIN) + D_{U6} = 27 + 1 + 1 = 29$$

The observability of the enable nodes depends on whether or not changes at the driver/receiver inputs are detected at the outputs. The driver/receiver inputs must be preset to one value, then toggled to the opposite value. The change will only propagate to the output only if the enable signal is "on". No change means the enable is "off".

$$\begin{aligned}
 CO(E1) &= CO(PIN) + CC0(D1) + CC1(D1) + D_{U1} \\
 &= 0 + 10 + 9 + 1 \\
 &= 20
 \end{aligned}$$

$$\begin{aligned}
 CO(E2) &= CO(D2) + CC0(PIN) + CC1(PIN) + D_{U2} \\
 &= 33 + 1 + 1 + 1 \\
 &= 36
 \end{aligned}$$

$$\begin{aligned}
 CO(E3) &= CO(PIN) + CC0(D3) + CC1(D3) + D_{U3} \\
 &= 0 + 22 + 13 + 1 \\
 &= 36
 \end{aligned}$$

$$\begin{aligned}
 CO(E4) &= CO(D4) + CC0(PIN) + CC1(PIN) + D_{U4} \\
 &= 18 + 1 + 1 + 1 \\
 &= 21
 \end{aligned}$$

$$\begin{aligned}
 CO(E5) &= CO(PIN) + CC0(D5) + CC1(D5) + D_{U5} \\
 &= 0 + 8 + 15 + 1 \\
 &= 24
 \end{aligned}$$

The final C/O values are shown in Fig. A2.3b. This is a greatly simplified example. One can expect to find many bi-directional I/O pins in a VLSI design, plus a few nodes which are imbedded deep inside the circuit which are not accessible directly from the device pins. The next example is somewhat more complicated because the C/O values of the bi-directional node are not known initially.

22
OF / DE

GLOSSARY

Bi-directionality : This refers to a circuit in which data is allowed to flow in two directions (but not both simultaneously) along its branches. Nodes connected to such branches are not specifically input or output nodes of cells or gates, but will be one or the other depending on the direction of signal flow.

CAMELOT : Stands for Computer Aided MEasure for Logic Testability, developed for the British Post Office by Bennetts, Maunder, and Robinson. Testability transfer functions are calculated for each submodule in a circuit in a manner very similar to TMEAS from Bell Labs. In CAMELOT, however, the observabilities take into account the controllabilities of "supporting" links, which are used to propagate a signal through the module.

Controllability : A function which represents the difficulty of accessing and setting a given node in the circuit to the desired value via one or more primary inputs, based on the minimum number of gates and nodes which must be assigned values, and the minimum number of sequential steps required to do so.

Controllability/Observability (C/O) : A term generally used (in this thesis) to refer to the values produced by the SCOAP testability analysis program.

Design For Testability (DFT) : A general name for many different techniques for enhancing the testability of a circuit design, from basic ad-hoc rules and guidelines to an all-encompassing design philosophy.

Device Under Test (DUT) : The device being tested, whether it is an IC, a board, or a system. The DUT is sometimes referred to as the "Circuit Under Test" (CUT), the "Unit Under Test" (UUT), or the "Object to be Tested" (OtBT).

Fault Simulation : A simulation in which faults are inserted into a logic circuit, and a test pattern is applied, in order to see how effectively that particular test pattern detects the presence of faults in the circuit.

Fault Tolerant Circuit (FTC) : A circuit with the capability to detect and correct internal faults, usually by means of redundant logic and majority voting modules.

Logic Simulation : A simulation in which a logic circuit is tested for timing and functionality (ie. does the circuit design perform the function which it was intended to).

LSSD : Acronym for Level Sensitive Scan Design. A systematic approach to design for testability in which a set of restricted design rules is applied so that the circuit reconfigures internal storage elements into shift registers for ease of testing, and to eliminate the dependence of testability on delay, rise, and fall times.

Observability : A function which represents the difficulty of accessing and observing the value of a given node via a primary output, based on the minimum number of gates and nodes which must be assigned values, and the minimum number of sequential steps required to do so.

Reconvergent fanout : A phenomenon in digital circuits where signal paths diverge from the inputs and converge again to one or only a few outputs.

SCOAP : Acronym for Scandia Laboratories Controllability / Observability Analysis Program. A testability analysis program intended for random logic ICs.

Self-Testing Circuits (STC) : Circuits which are able to detect and signal the presence of internal faults, but not able to internally correct the error which results. STC design reduces the demands on the testing equipment hardware and software, and the skill and experience of the operator.

Testability : A measure of the relative ease of testing a circuit which, for digital circuits, is related to the length of the test pattern required to thoroughly verify the functionality of the network.

Testability Analysis (TA) : A means of estimating the ease with which a given circuit can be tested.

TESTSCREEN : A testability analysis program developed by Kovijanic at Sperry Research which calculates circuit node controllabilities and observabilities very much like SCOAP. There are differences in the way cells are modelled.

TMEAS : Acronym for Testability MEASuring program developed by Grason at Bell Labs. Testability transfer functions are calculated for each cell or module in the circuit, based on signal probabilities. This method of testability analysis is hierarchical, that is transfer functions of submodules can be used to find transfer functions of modules, etc. up to the system level.

BIBLIOGRAPHY

- [1] HEALY, J.T.; "Automated testing and evaluation of digital integrated circuits"; Reston Publ. Co. Inc.: Reston, VA, (1981).
- [2] BREUER, M.A., FRIEDMAN, A.D.; "Diagnosis and reliable design of digital systems"; Computer Sciences Press Inc.: Rockville, MD, (1976).
- [3] LIBERTY, S.R., SAEKS, R. (Editors); "Rational fault analysis"; Marcell Dekker Inc.: NYC, (1974).
- [4] MUEHLDOERF, E.I., SAVKAR, A.D.; "LSI logic testing - an overview"; IEEE Trans. on Computers, Vol. C-30, No. 1, (Jan 1981), p1-17.
- [5] LAI, L. K.-W., SIEWIOREK, D.P.; "Testing of digital systems"; Proc. of the IEEE, Vol. 69, No. 10, (Oct. 1981), p1321-1333.
- [6] PARKER, K.P., WILLIAMS, T.W.; "Design for testability - a survey"; IEEE Trans. on Computers, Vol. C-31, No. 1, (Jan 1982), p2-15.
- [7] SZYGENDA, S.A.; "TEGAS2 - Anatomy of a general purpose test generation and simulation system for digital logic"; Proc. 9th Design Automation Conf., (June 1972), p116-127.
- [8] ROMBEEK, H., WILCOX, P.; "F/LOGIC - An interactive fault and logic simulator for digital circuits"; Proc. 13th Design Automation Conf., (June 1976), p68-73.
- [9] GOLDSTEIN, L.H.; "Controllability/Observability analysis of digital Circuits"; IEEE Trans. on Circuits & Systems, Vol. CAS-26, No. 9, (Sept 1979), p685-693.
- [10] GOLDSTEIN, L.H., THIGPEN, E.L.; "SCOAP: Sandia Controllability / Observability analysis program"; Proc. 17th Design Automation Conf., (June 1980), p190-196.

- [11] KOVIJANIC, P.G.; "Testability Analysis"; Digest 1979 IEEE Test Conf., (1979), p310-316.
- [12] KOVIJANIC, P.G.; "Single testability figure of merit"; Digest 1981 IEEE Test Conf., (1981), p521-529.
- [13] GRASON, J., STEPHENSON, J.E.; "A testability measure for register transfer level digital circuits"; Proc. 6th Fault Tolerant Computing Symp., (June 1976), p101-107.
- [14] GRASON, J.; "TMEAS, a testability measurement program"; Proc. 16th Design Automation Conf., (June 1979), p156-161.
- [15] BENNETTS, R.G., MAUNDER, C.M., ROBINSON, G.D.; "CAMELOT: A computer aided measure for logic testability"; Proc. IEEE Intl. Conf. on Circuits and Computers, (Oct. 1980), p1162-1165.
- [16] BENNETTS, R.G., MAUNDER, C.M., ROBINSON, G.D.; "CAMELOT: A computer aided measure for logic testability"; Proc. Workshop on Self-Diagnosis and Fault Tolerance, Tübingen Univ., (1981), p30-45.
- [17] GOEL, D.K., McDERMOTT, R.M.; "An interactive testability analysis program - ITAP"; Proc. 19th Design Automation Conf., (June 1982), p581-586.
- [18] EICHELBERGER, E.B., WILLIAMS, T.W.; "A logic design structure for LSI testability"; Proc. 14th Design Automation Conf., (June 1977), p462-468.
- [19] BUEHLER, M.G.; SIEVERS, M.W.; "Off-line, built-in test techniques for VLSI circuits"; Computer Vol. 15, No. 6, (June 1982), p69-82.
- [20] KOENEMANN, B., MUCHA, J., ZWIEHOFF, G.; "Built-in logic block observation techniques"; Digest 1979 IEEE Test Conf., (Oct. 1979), p37-41.
- [21] TSUI, F.F.; "In-situ testability design (ISTD) - a new approach for testing high-speed LSI/VLSI logic"; Proc. of the IEEE, Vol. 70, No. 1, (Jan 1982), p59-78.

- [22] TURINO, J.; "Design for Testability"; Logical Solutions Inc.: Campbell, CA, (1979).
- [23] FUNATSU, S., KAWAI, M., TAKASAKI, S., YAMADA, A.; "A calculus of testability measure at the functional level"; Digest 1981 IEEE Test Conf., (1981), p95-101.
- [24] GRASON, J.; "Automatic test point selection using TMEAS"; 1982 Design For Testability Workshop, (Apr. 22, 1982), oral presentation.
- [25] HESS, R.D.; "Testability analysis: An alternative to structured design for testability"; VLSI Design, Vol. 3, No. 2; (Mar/Apr 1982), p22-28.
- [26] SEGERS, M.T.M.R.; "The impact of testing on VLSI design methods"; IEEE Jour. Solid State Circuits, Vol. SC-17, No. 3, (June 1982), p481-486.

END

05109184.

FIN