

Fault Characterization and Testability Considerations in Multi-Valued Logic Circuits

by

Maher Mohammed Mahmoud Al-Sharif

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER ENGINEERING

December, 1998

INFORMATION TO USERS

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

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

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

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

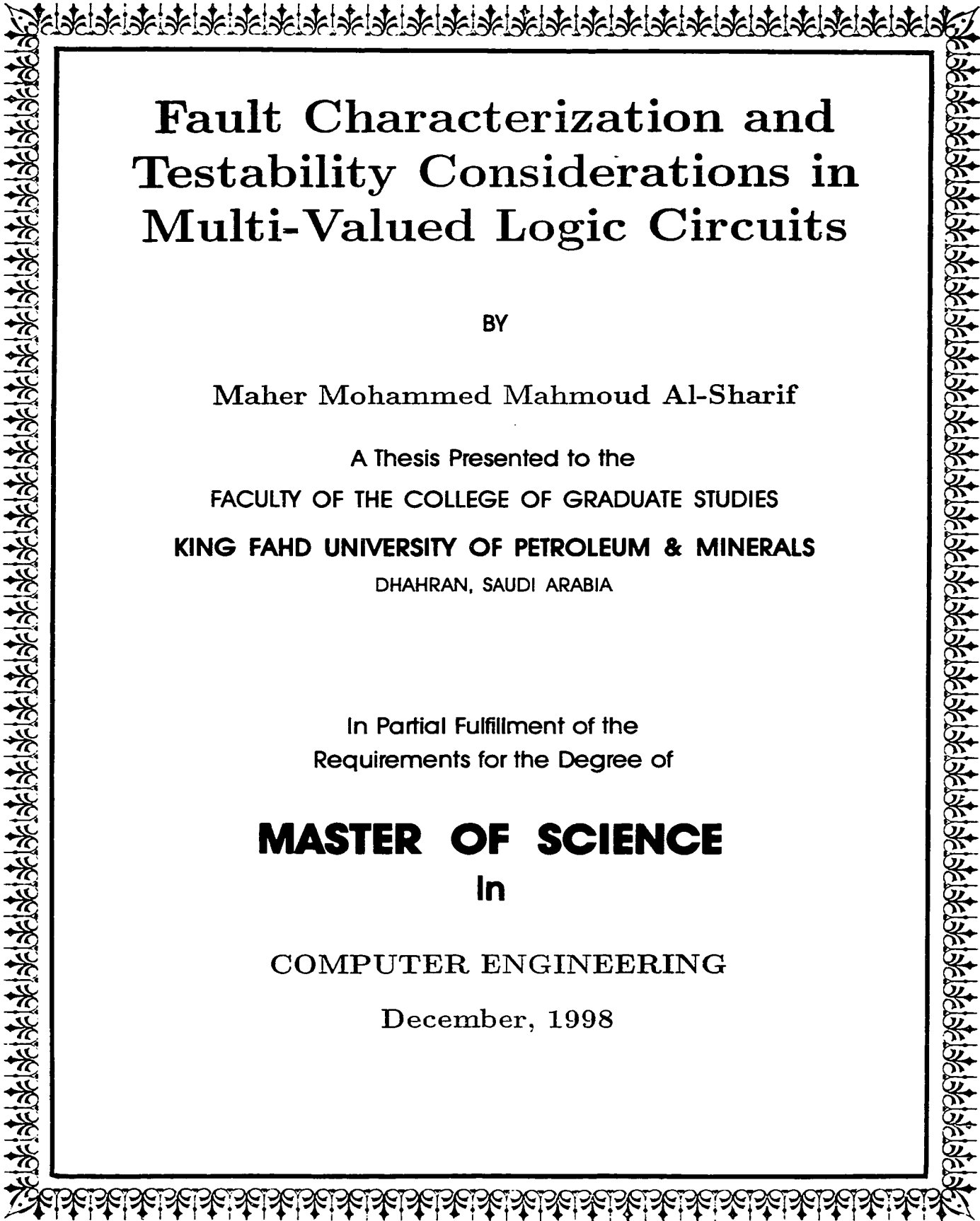
ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI[®]

NOTE TO USERS

This reproduction is the best copy available.

UMI[®]



Fault Characterization and Testability Considerations in Multi-Valued Logic Circuits

BY

Maher Mohammed Mahmoud Al-Sharif

A Thesis Presented to the
FACULTY OF THE COLLEGE OF GRADUATE STUDIES
KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE
In

COMPUTER ENGINEERING

December, 1998

UMI Number: 1403696

UMI[®]

UMI Microform 1403696

Copyright 2001 by Bell & Howell Information and Learning Company.

All rights reserved. This microform edition is protected against
unauthorized copying under Title 17, United States Code.

Bell & Howell Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS
DHAHRAN 31261, SAUDI ARABIA

COLLEGE OF GRADUATE STUDIES

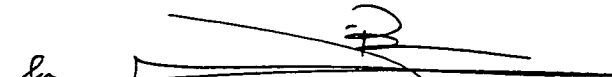
This thesis written by


Maher Mohammed Al-Sharif

under the direction of his Thesis Advisor and approved by his Thesis Committee,
has been presented to and accepted by the Dean of the College of Graduate Studies,
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER ENGINEERING


Thesis Committee:


Dr. Mohammed Osman (Chairman)

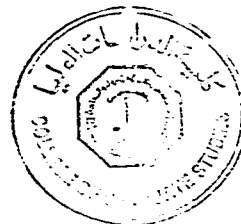
99

Dr. Mostafa Abd - El - Barr (Co - Chairman)


Dr. Alaaeldin Amin (Member)


Department Chairman
Abdullah Almjel


Dean, College of Graduate Studies

30-1-99
Date



To my Family

and to

Bait Tema people

Acknowledgments

All praise be to Allah for his limitless help and guidance. Peace and blessings of Allah be upon His prophet Muhammad.

Acknowledgment is due to King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, for the generous support for this research. I would like to express my profound gratitude and appreciation to my advisors, Dr. Mohammed Osman and Dr. Mostafa Abd-El-Barr for their support and patience throughout this thesis. Their continuous support and encouragement can never be forgotten. I would like to thank Dr. Alaaeldin Amin for his consistent support and valuable discussions.

I also wish to thank faculty, graduate students, and the staff members of the Computer Engineering Department for their support. The encouragement and good wishes of my friends, Abdallah Rayhan, Maher Mutlaq, Raed Basha, Jamil Sangrar, Ahmad Ashadawi, Samer Saed, Kalil Balawi, Abdul Rahman Abdul Raheem, Firas Maadi, Ihab Al-Saqqa and Mohammed Bawadi are also worthy acknowledged. Special thanks must be given to my father, the family and my wife for their encouragement and moral support.

Contents

List of Tables	viii
List of Figures	x
Abstract (English)	xiii
Abstract (Arabic)	xv
1 Introduction	1
1.1 General	1
1.2 Motivation	2
1.3 Objectives	3
1.4 Thesis Outline	4
2 Background Material	6
2.1 Functionally Complete Sets of MVL Operators	7
2.1.1 Definitions	8

2.1.2	CMOS Circuit Realizations	9
2.2	Testability Issues in MVL Circuits	16
2.2.1	K-enabled Gates and Fault Propagation	17
3	Literature Review	22
4	Physical Defect Modeling and Simulation	31
4.1	Physical Defects	31
4.1.1	Physical Defects & Faults	33
4.1.2	Spot Defects	34
4.2	Fault Characterization	36
4.2.1	Device Level Fault Characterization	36
4.2.2	Layout Level Fault Characterization	39
4.2.3	Comparison	42
5	Fault Characterization of the MVL Set	43
5.1	The Approach	44
5.2	Fault Characterization of the Literal	48
5.3	Fault Characterization of the Complement of Literal	57
5.4	Fault Characterization of the Cycle	64
5.5	Fault Characterization of the <i>tSum</i>	71
5.6	Fault Characterization of the <i>Min</i>	74

5.7	Fault Characterization Summary	80
6	Testing of the MVL Set	81
6.1	Testing of the Literal	82
6.2	Testing of the Complement of Literal	89
6.3	Testing of the Cycle	89
6.4	Testing of the <i>tSum</i>	95
6.5	Testing of the <i>Min</i>	101
6.6	Peculiar Faults Testing and Fault Coverage	108
6.6.1	Oscillations Fault	109
6.6.2	A Soft Error	111
6.6.3	3* Faults	111
7	Design for Testability	116
7.1	An Application	117
8	Discussion and Conclusions	122
8.1	Summary of the Main Results of the Thesis	123
8.2	Future Work	124
	Bibliography	126
	Vita	131

List of Tables

5.1	Fault characterization of the literal circuit.	58
5.2	Fault characterization of the complement of literal circuit.	65
5.3	Fault characterization of the cycle circuit.	72
5.4	Fault characterization of the <i>tSum</i> circuit.	76
5.5	Fault characterization of the <i>min</i> circuit.	79
5.6	Summery of the characterized fault categories.	80
6.1	Exhaustive test list for the literal circuit	86
6.2	Fault coverage tables for $k^1[x]^1$	88
6.3	Literal tests.	89
6.4	Minimal test sets/sequences for the literal.	90
6.5	Exhaustive test list for the complement of literal.	91
6.6	Complement of literal tests.	93
6.7	Minimal test sets/sequences for the complement of literal.	94
6.8	Exhaustive test list for the cycle.	96

6.9	Cycle tests.	98
6.10	Minimal test sets/sequences for the cycle.	98
6.11	Exhaustive test list for the sequential faults in the <i>tSum</i>	99
6.12	Exhaustive test list for the functional faults in the <i>tSum</i>	100
6.13	<i>tSum</i> tests.	100
6.14	Exhaustive test list for the <i>min</i> , sequential part 1.	103
6.15	Exhaustive test list for the <i>min</i> , sequential part 2.	104
6.16	Exhaustive test list (in terms of I2) for the <i>min</i> , sequential part 3. . .	105
6.17	I2 values and the corresponding $\langle x, y \rangle$ input values.	105
6.18	Exhaustive test list for the <i>min</i> , sequential part 3.	106
6.19	Exhaustive test list for the <i>min</i> , functional part 1.	106
6.20	Exhaustive test list for the <i>min</i> , functional part 2.	107
6.21	Minimal test sets/sequences for the <i>min</i>	108
6.22	Fault coverage figures and 3* testing.	115
7.1	Truth table for an MVL full adder.	119
8.1	Summery of the characterized fault categories.	123

List of Figures

2.1	Basic circuit elements for CMCL.	11
2.2	Circuit realizing <i>min</i> operator. (a): input currents are sinking, (b): input currents are sourcing	12
2.3	Transient-time response of the <i>min</i> operator.	12
2.4	Circuit realization of <i>tsum</i>	13
2.5	Transient-time response of the <i>tsum</i> operator.	13
2.6	Realization of a <i>literal</i> $k[\{x\}^b]$ operator.	14
2.7	Realization of a <i>complement of literal</i> $\overline{k[\{x\}^b]}$ operator.	14
2.8	Transient-time response of $3[\{x\}^2]$ and $\overline{2[\{x\}^1]}$ operators.	15
2.9	Circuit realization of the <i>cycle</i> operator.	15
2.10	Transient-time response of the <i>cycle</i> operator.	16
2.11	k-enabled gates.	18
2.12	Fully enabled gate.	20
2.13	1-enabled gate.	20
3.1	Different D-chains propagation.	24

3.2	Event driven fault simulation.	28
3.3	Example for <i>free vertical crossover</i> genetic operator.	30
4.1	Spot defects overlapping two or more wires.	34
4.2	Open and short faults and their models.	39
5.1	Open and short faults and their models.	45
5.2	Literal transistor diagram.	50
5.3	Response of the literal, $3^1[x]^1$	51
5.4	Node A current and voltage.	52
5.5	Node B current and voltage.	53
5.6	Response of the circuit for literal $3^1[x]^1$ in the presence of Slgs.	54
5.7	Node A current and voltage in the presence of Slgs.	55
5.8	Node B current and voltage in the presence of Slgs.	56
5.9	Complement of Literal transistor diagram.	60
5.10	Response of the complement of literal, $\overline{2^2[x]^3}$	61
5.11	Response of the complement of literal, $\overline{3^2[x]^2}$	62
5.12	Response of the complement of literal, $\overline{3^2[x]^2}$ in the presence O10d.	63
5.13	Cycle transistor diagram.	67
5.14	Response of the cycle, X^{-1}	68
5.15	Response of the cycle, X^{-2}	69

5.16	Response of the cycle, $X^{\leftarrow 2}$ in the presence of O13d.	70
5.17	<i>tSum</i> transistor diagram.	74
5.18	Response of a <i>tSum</i> circuit.	75
5.19	Response of a <i>tSum</i> circuit in the presence of O3d.	76
5.20	<i>Min</i> transistor diagram.	77
5.21	Response of a <i>min</i> circuit.	78
5.22	Response of a <i>min</i> circuit in the presence of O5s.	79
6.1	Fault S6gd charging and discharging paths (the literal circuit).	110
6.2	Fault S3gd charging and discharging paths (the <i>tSum</i> circuit).	110
7.1	A 4-valued adder.	120
7.2	A testable 4-valued adder.	121

Thesis Abstract

Name: Maher Mohammed Al-Sharif
Title: Fault Characterization and Testability Considerations
In Multi-Valued Logic Circuits
Major Field: Computer Engineering
Date of Degree: December, 1998

With the growing interest and the emergence of various implementations of Multiple-Valued Logic circuits (MVL), testability issues of these circuits are becoming crucial. Fault characterization is aimed at finding fault models that best describe faults expected to occur in a given class of circuits or technology. It can be performed either at the layout or the device level representations. Layout level techniques start by inserting defects in the layout of a fault free circuit. The resultant circuit is then extracted from the layout and simulated to study its behavior under the inserted fault. Device level techniques start by inserting shorts and opens directly to the device-level-representation of the fault free circuit. The behavior of the resultant circuit is then studied. In this thesis, we aim to characterize faults in CMOS MVL circuits at the device level. For this purpose, a functionally complete set of MVL operators will be used. The set has been implemented using existing standard binary CMOS technology. This enables us to use the same techniques used for standard binary CMOS. Fault categories in MVL circuits and recommendations for testability will be given.

Key Words: MVL circuits, Testing, Fault characterization, Fault modeling, Testability considerations, MVL testing

Master of Science Degree

King Fahd University of Petroleum and Minerals
Dhahran, Saudi Arabia

December, 1998

خلاصة الرسالة

اسم الطالب الكامل: ماهر محمد محمود الشريف

عنوان الدراسة: تحديد الأخطاء واعتبارات الاختبارية

في الدوائر الرقمية المتعددة القيم المنطقية

التخصص: هندسة الحاسب الآلي

تاريخ الشهادة: ديسمبر (كانون أول) ١٩٩٨م - رمضان ١٤١٩هـ

مع الاهتمام المتزايد والوصول إلى العديد من بناءات الدوائر الرقمية المتعددة القيم المنطقية، تصبح أمور الاختبارية لهذه الدوائر أكثر أهمية. اختبار هذه الدوائر يعتبر أكثر تعقيداً من اختبار الدوائر ثنائية القيم المنطقية. يعزى هذا للعدد الأكبر من القيم المنطقية التي يستطيع حملها أي خط في الدوائر المتعددة القيم المنطقية. وهذا يؤدي بدوره إلى ظهور بعض الظواهر الجديدة مثل البوابات الجزئية التمكين. تحديد الأخطاء هو خطوة أساسية مبكرة في عملية تكوين الاختبار ويعني بإيجاد نموذج للأخطاء بحيث يتضمن الأخطاء المتوقع حدوثها في أي تكنولوجيا معطاة. تحديد الأخطاء يمكن عمله على مستوي الهيكلية أو على مستوى الترانسسور. الطرق التي تعمل على مستوي الهيكلية تبدأ بإدخال الأخطاء في هيكل الدائرة الصحيحة. يتم بعد ذلك استخراج تمثيل الترانسسور لهذه الدائرة مع الخطأ المدخل سابقاً ومن ثم محاكاتها لمعرفة تصرفها في وجود الخطأ. أما الطرق التي تعمل على مستوي الترانسسور فهي تبدأ بإدخال توصيلات زائدة أو فراغات مباشرة إلى تمثيل الترانسسور للدائرة الصحيحة. على سبيل المثال، يمكن إدخال توصيلات زائدة بين طرفين من أطراف أحد الترانسسورات ويتم بعد ذلك محاكاة الدائرة الناتجة لدراسة تصرفها في وجود هذا الخطأ. الدراسات على مستوي الهيكلية أو مستوى الترانسسور على تقنيات سيموس وبايسموس أثبتت أن تمثيل الأخطاء بنموذج التعلق بالخطأ ليس دقيقاً كفاية لتمثيل الأخطاء الفيزيائية التي تحدث في الواقع. في هذه الرسالة عرضنا هو تحديد الأخطاء في الدوائر الرقمية المتعددة القيم المصنوعة بتقنية سيموس على مستوي تمثيل الترانسسور. لهذا الغرض، تم اختيار

مجموعة من العمليات المتكاملة ادائياً من الدوائر الرقمية المتعددة القيم المنطقية. هذه العمليات تم بناؤها بواسطة تقنية سيموس مما يمكننا من تحديد الأخطاء فيها باستخدام الطرق المعروفة لهذه التقنية. تصنيف الأخطاء في الدوائر الرقمية المتعددة القيم المنطقية وإرشادات للاختبارية سوف تقدم في هذا البحث.

درجة الماجستير في العلوم

جامعة الملك فهد للبترول والمعادن

الظهران، المملكة العربية السعودية

ديسمبر (كانون أول) ١٩٩٨ م - رمضان ١٤١٩ هـ

Chapter 1

Introduction

1.1 General

In MVL circuits, signal lines are not restricted to two values. Theoretically, they can carry any desired number of logic values. This increases the amount of information and reduces the space requirements, two desirable features in very large scale integration (VLSI) implementation. Until recently, practical implementations of MVL circuits were not possible due to technology limitations. It has been recently possible to implement MVL circuits using the available standard binary CMOS technology. This resulted in various MVL implementations either as stand alone circuits, e. g. multiplier chips [22], or as modules used in larger binary circuits. Intel, for example,

used MVL in the ROM part of its 8087 coprocessor and largely reduced the space requirements making it possible to fit the circuit in a standard chip size [28].

1.2 Motivation

With the increasing number of MVL implementations, testing such circuits is becoming crucial. It is more involved to test an MVL circuit as compared to its binary counterpart. This is due to the increased number of logic values and the need to distinguish and eliminate an increased number of possible logic choices. However, during the test generation process, there exists some flexibility since decisions to assign a certain value to a line are not restricted to 0 or 1 only.

Previous work in testing MVL circuits includes generalizing binary testing techniques to the MVL case. Ajab Noor and Abd-El-Barr [1], for example, used Boolean Difference-like method in testing for stuck-at faults in MVL circuits. Spillman and Su [39] modified the D-algorithm to test MVL circuits. Tabakow [42] introduced a generalized D-algebra for error propagation and detection. Other work used new concepts, like the partial sensitivity for gates, found in MVL and formalized them to be used in test generation. For example, Dubrova et. al. [8] used a full sensitivity concept and formalized it to generate expressions to be used in test generation.

Previous work on MVL testing has been primarily based on the stuck-at fault model. The adequacy of this model was questioned by various studies (see Chapter 4). Also, fault characterization of MVL circuits has remained unattended so far. For this reason, and due to the increased complexity of test generation of MVL circuits, it is very important to recognize the actual type of faults expected to occur in MVL implementations. This characterization becomes then an important input to any test generation algorithm.

1.3 Objectives

The aim of this research work is to characterize faults in MVL circuits. For this purpose, a functionally complete set of MVL operators is selected. The set is described in Chapter 2. This set is not unique and is not minimal, but it is more efficient in realizing MVL functions [20, 21]. One advantage of this set is that it was implemented using standard binary CMOS technology. This enables us to use the same fault characterization techniques used in binary CMOS in order to characterize CMOS MVL circuits. Fault characterization techniques can be conducted at two main levels, namely the layout level or the device level. In the first, a defect is inserted in the layout of a fault free circuit. The new circuit, with the inserted fault, is then extracted from the layout. Next, the resultant circuit is simulated

to study its behavior under the inserted fault. In the second approach, shorts and opens are inserted directly to the device-level representation of the fault free circuit. For example, a short can be inserted between two terminals of a transistor. Then, the behavior of the resultant circuit is studied. In this work, fault characterization in the considered set of MVL operators, is conducted at the **device level**.

1.4 Thesis Outline

The rest of this thesis is organized as follows. Chapter 2 introduces some background material. A functionally complete set of MVL operators is introduced. The second part of Chapter 2 introduces some new phenomena found in MVL and their effect on the test generation process. A survey of previous MVL testing work is given in Chapter 3. Chapter 4 surveys studies on the actual physical defects found in VLSI implementations. Such studies form the basis of any fault characterization technique. Techniques used in fault characterization for binary CMOS and BiCMOS are also detailed in Chapter 4. Chapter 5 describes the simulations conducted to characterize the faults. Fault categories found for the operators of the MVL set are given. Further, Chapter 6 illustrates the use of these results to find the valid test vectors for all faults found in an operator circuit by exhaustive simulation. Chapter 7 presents Design for Testability (DFT) recommendations and applications.

Discussion and conclusions are given in Chapter 8 which ends with some possible future work.

Chapter 2

Background Material

In MVL circuits, signal lines are allowed to carry more than two logic values. A line in a 4-valued logic system can represent 4 different logic levels: 0, 1, 2 and 3. To represent the same amount of information in binary logic, will require two lines. This reduction in the number of lines introduces a potential for reducing the wiring interconnect, a very desirable feature needed in very large scale integration (VLSI) implementation. Another advantage of MVL is the increased functionality. For example, using one binary variable, 4 (2^{2^1}) different functions can be realized, whereas using a four-valued variable allows the realization of 256 (4^{4^1}) different functions. In general, there are r^{r^n} different functions in an r -valued, n -variable system [20, 21]. Although some MVL algebras were developed as early as Boolean algebra, practical

implementations were not possible due to technology limitations. Recently, MVL gained increasing interest since it became possible to implement MVL circuits using available technologies. Implementation of MVL circuits using the already available binary CMOS process have been reported [20, 21].

Basically there are two ways for using MVL [45]. One, is to use interconnect lines carrying MVL signals between binary devices and gates. Encoding from binary to MVL and decoding from MVL to binary circuits will be needed in this scheme. The second method is to design the whole circuit or some modules (devices and the interconnect) using MVL.

2.1 Functionally Complete Sets of MVL Operators

A functionally complete set of MVL operators is a set of MVL operators capable of realizing any arbitrary MVL function. In this section, a functionally complete set of MVL operators is introduced. The set is reported in the work of Jain et. al. [20, 21]. It is implemented using CMOS and consists of six operators: *literal*, *cycle*, *complement of literal*, *complement of cycle*, *min* and *tsum*. Different current values are used to represent the MVL levels. In this implementation, 4-valued logic values

are used. Internal threshold circuit elements generate binary voltage signals which control switches that realize appropriate current levels. The functionality of the set implementation is verified by HSPICE simulation [20].

The introduced set is not unique. There are other MVL sets of operators which are also functionally complete. Moreover, the set is not minimal. For example, *literal*, *min* & *tsum* are functionally complete. However, the introduced set is shown to be more efficient in realizing MVL functions in terms of the number of operators used and the overall required circuits [21].

2.1.1 Definitions

Let $R = \{0, 1, 2, \dots, r - 1\}$ be the set of logic values for an r -valued logic where r is the radix. Assume also that $a, b, \in R$, $a \leq b$ and $k \in \{1, 2, \dots, r - 1\}$. The following MVL operations are defined.

1. *min* (minimum operator) is defined as

$$\min(a_1, a_2, \dots, a_n) = a_1 \bullet a_2 \bullet \dots \bullet a_n = \text{minimum of } (a_1, a_2, \dots, a_n) \text{ where}$$

$$a_1, a_2, \dots, a_n \in R. \text{ For example, } \min(1, 2, 3) = 1.$$

2. *tsum* (truncated sum) operator is defined as

$$\textit{tsum}(a_1, a_2, \dots, a_n) = a_1 \oplus a_2 \oplus \dots \oplus a_n = \min(a_1 + a_2 + \dots + a_n, r - 1). \text{ For}$$

example, for $r = 4$, $a_1 = 2$, $a_2 = 3$, and $a_3 = 1$, $tsum(2, 3, 1) = \min(2 + 3 + 1, r-1) = \min(6, 3) = 3$.

3. *complement* of a logic level l is defined as

$$\bar{l} = (r - 1) - l. \text{ For example, for } r = 4 \text{ and } l = 1, \bar{l} = 3 - 1 = 2.$$

4. *literal* of an MVL variable x is defined as

$${}^a\{x\}^b = \begin{cases} r-1 & \text{if } a \leq x \leq b \\ 0 & \text{otherwise} \end{cases}$$

A weighted literal, $k[{}^a\{x\}^b]$, is defined as

$$k[{}^a\{x\}^b] = \begin{cases} k & \text{if } a \leq x \leq b \\ 0 & \text{otherwise} \end{cases}$$

For example, for $r = 4$, $2[{}^1\{x\}^2]$ is equal to 2 if $x \in \{1, 2\}$ and 0 if $x \in \{0, 3\}$

5. *cycle* operator. A clockwise *cycle* is defined as

$\overrightarrow{x}^m = (x + m) \bmod r$ where $m \in R$. A counter clockwise *cycle* is defined as

$\overleftarrow{x}^m = (x - m) \bmod r$. For example, for $r = 4$, if $x = \langle 0123 \rangle$ then $\overrightarrow{x}^1 = \langle 1230 \rangle$.

2.1.2 CMOS Circuit Realizations

The circuit realization, presented in [20], for the above mentioned operators uses both current mode CMOS logic (CMCL) and voltage mode CMOS logic (VMCL)

circuit elements. CMCL elements are used to represent MVL levels while binary signals are represented by VMCL elements.

Figure 2.1 shows the basic circuit elements of CMCL which are used in the realization of the above operators. Figure 2.2 shows the realization of the *min* operator. Its transient-time response is shown in Figure 2.3 (scanned from [20] with permission). The *tsum* realization is shown in Figure 2.4 and the transient-time response in Figure 2.5 (scanned from [20] with permission). *Literal* and *complement of literal* operators are shown in Figures 2.6 and 2.7 respectively. Figure 2.8 shows the transient-time responses of $3[{}^1\{x\}^2]$ and $\overline{2[{}^1\{x\}^1]}$ (scanned from [20] with permission). Figure 2.9 shows the cycle realization. Finally, Figure 2.10 (scanned from [20] with permission) shows the (\overleftarrow{x}^2) cycle operator transient-time response.

The above realizations will be used in this work to study the possible types of faults that can occur at the end of a production line. The following sections will review fault characterization studies conducted on typical CMOS circuits. The methods used to model possible faults in those studies will be applicable to the above MVL realizations since the same binary CMOS technology is used in both. Some general MVL testability issues are presented in the next subsection.

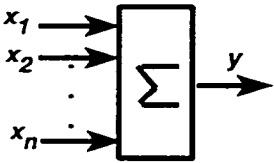
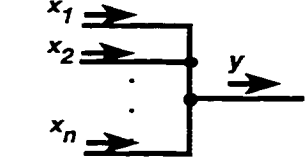
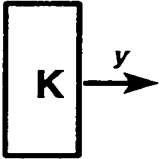
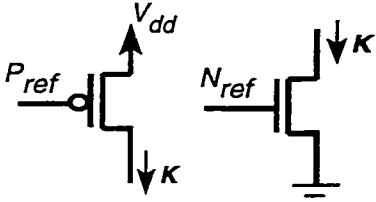
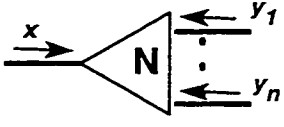
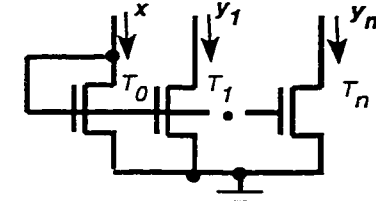
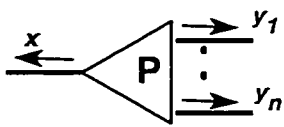
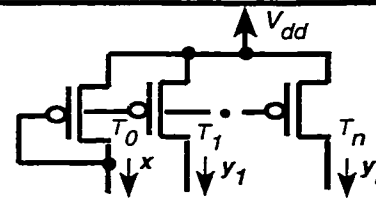
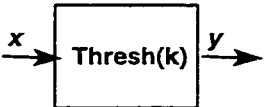
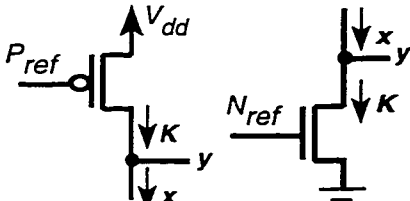
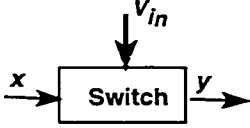
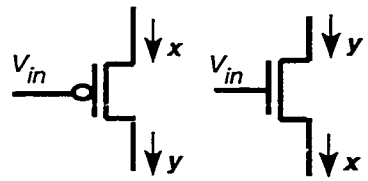
Name	Logic Operation	Symbol	Circuit Realization
Sum	$y = x_1 + x_2 + \dots + x_n$		
Constant	$y = k$		
NMOS and PMOS Current Mirror	$y_i = a_i x$ for $i = 1, 2, \dots, n$ a_i : scale factor		
			
P-type and N-type Threshold	<u>For N-type</u> if $(x \geq k)$ then $y = \text{binary high}$ else $y = \text{binary low}$		
P-type and N-type Switch	if (Switch is ON) then $y = x$ else (Switch is OFF) $y = 0$ (no current)		

Figure 2.1: Basic circuit elements for CMCL.

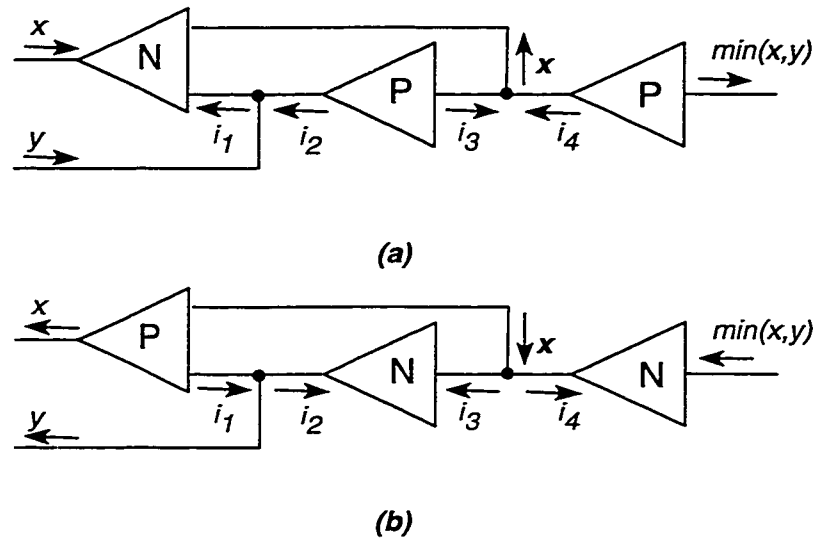


Figure 2.2: Circuit realizing \min operator. (a): input currents are sinking, (b): input currents are sourcing

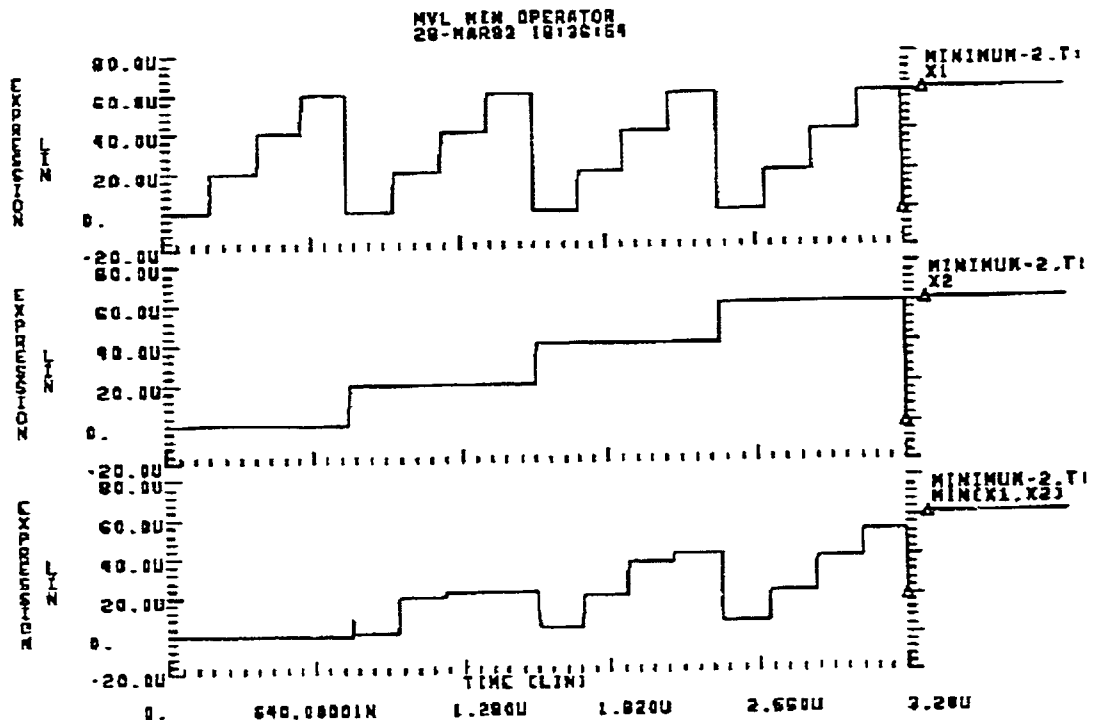
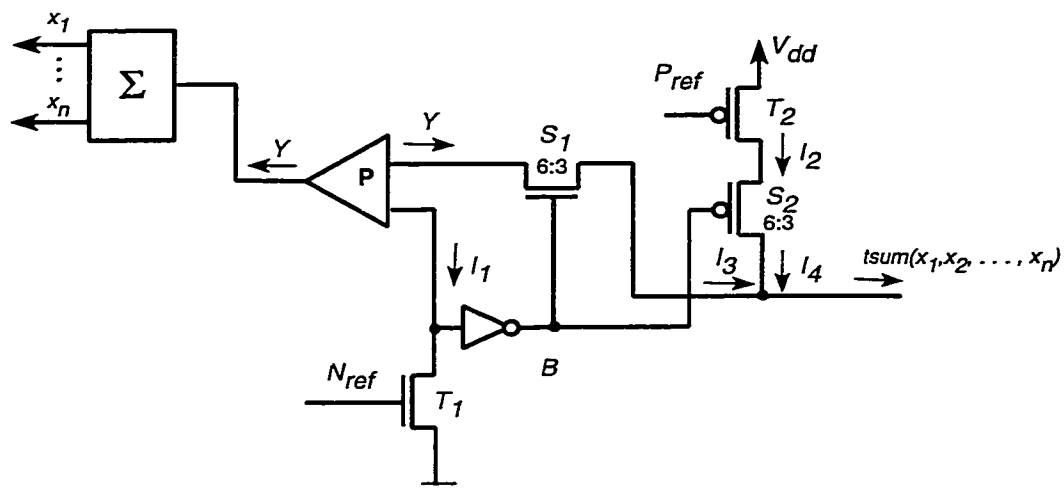
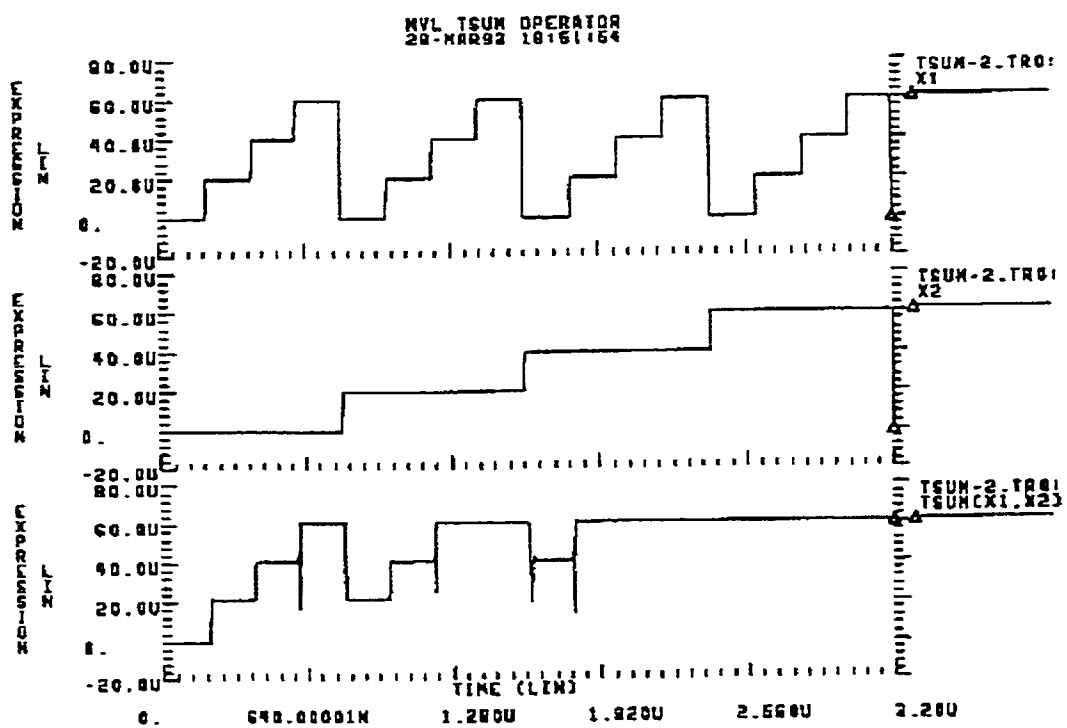


Figure 2.3: Transient-time response of the \min operator.

Figure 2.4: Circuit realization of *tsum*.Figure 2.5: Transient-time response of the *tsum* operator.

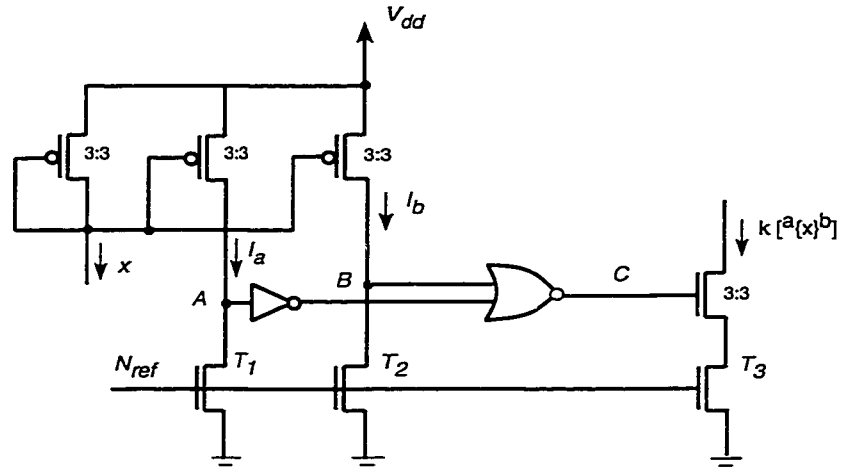


Figure 2.6: Realization of a *literal* $k[\{x\}^b]$ operator.

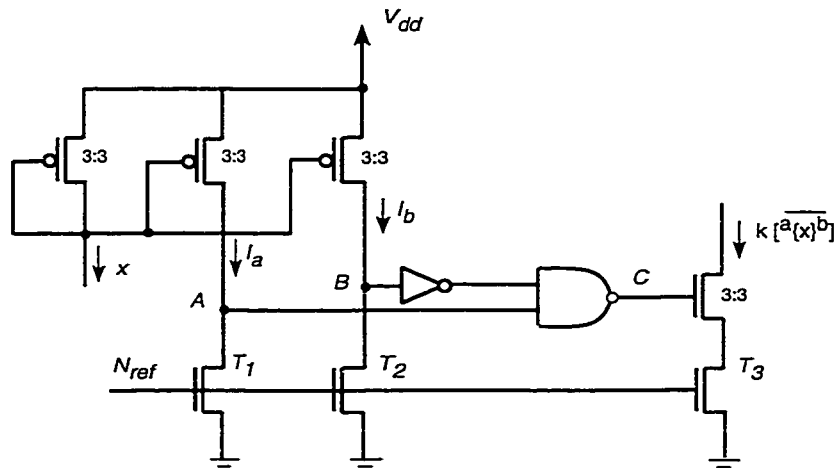


Figure 2.7: Realization of a *complement of literal* $k[\overline{\{x\}^b}]$ operator.

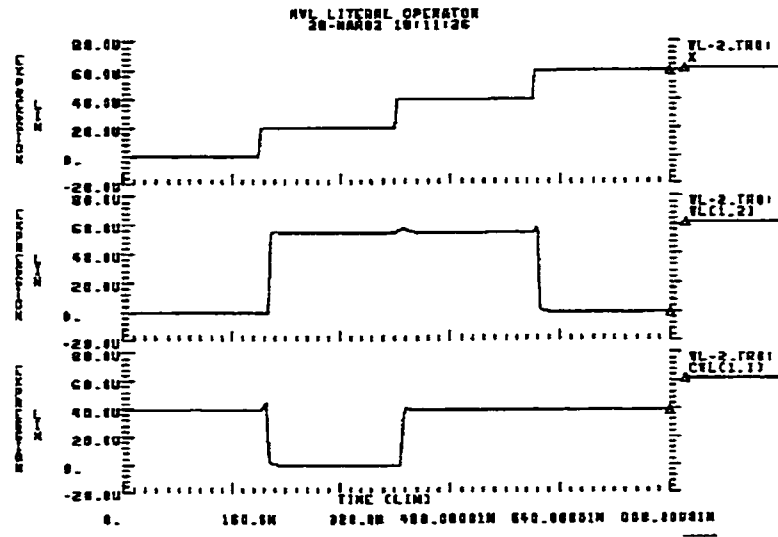


Figure 2.8: Transient-time response of $3[{}^1\{x\}^2]$ and $\overline{2[{}^1\{x\}^1]}$ operators.

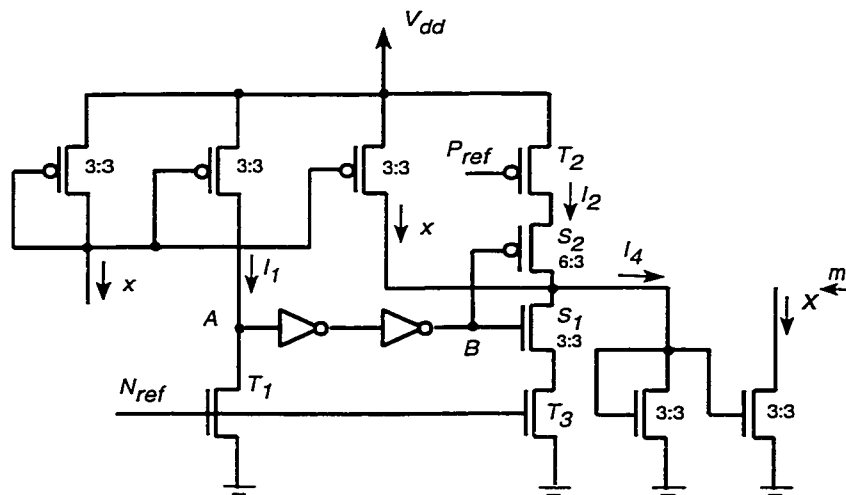


Figure 2.9: Circuit realization of the *cycle* operator.

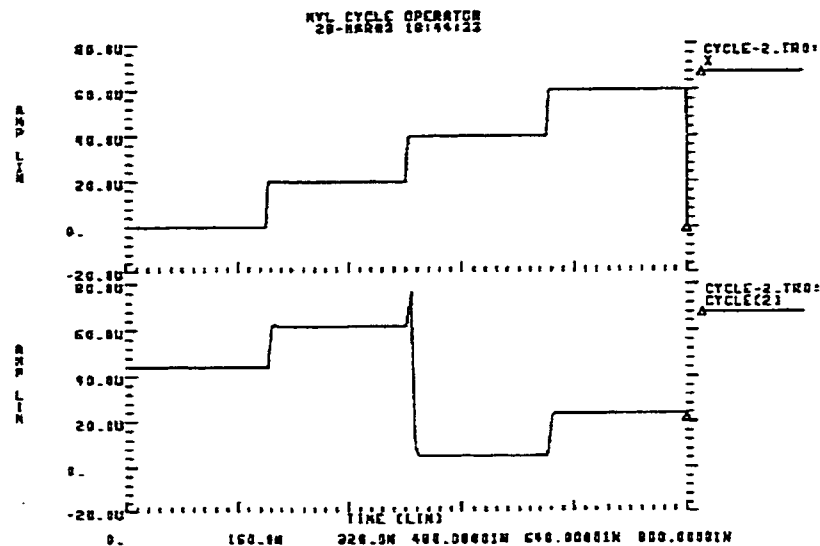


Figure 2.10: Transient-time response of the *cycle* operator.

2.2 Testability Issues in MVL Circuits

For a given line in a MVL network, the number of possible values that can be assigned is more than those assigned to a binary line. This complicates the test generation process as compared to the binary case. This is due to the increased number of logic values and the need to distinguish and eliminate an increased number of logic choices. However, during the test generation process, there exists some flexibility since decisions to assign a certain value to a line are not restricted to 0 or 1 only.

In addition, new concepts related to the MVL arise. At any given time, an MVL gate can be either partially or fully enabled. A gate is fully enabled along one of its inputs if the gate's output is sensitive to any change on that input. This can be

achieved by setting other inputs to appropriate values. A 2-input binary AND gate can be fully enabled (sensitive) along one of its inputs if the other input, call it the fixed input, is set to 1. So any change on the free input can be sensed at the output. If, on the other hand, the fixed input was assigned to 0, the gate is disabled and changes on the free input have no effect on the output, which will be always 0.

The same concept (enabled or disabled gate) can be generalized to the 2-input MVL *min* gate by substituting 0s, in binary, with 0s, in MVL, and 1s with $r-1$. However, if the fixed input is set to k , where $0 < k < r - 1$, the *min* gate is partially enabled (k -enabled). It will be sensitive to some values and disabled for some others. This case does not exist in binary.

To illustrate the above concept, consider the 4-valued, 2-input *min* gate shown in Figure 2.11a where the fixed input is assigned to k . All inputs greater than or equal to k are indistinguishable at the output. If $k = 2$ (Figure 2.11b), the gate is sensitive only to 0 and 1. The output is always 2 for the rest of input values. So, input changes for values ≥ 2 have no distinguishable effect on the output.

2.2.1 K-enabled Gates and Fault Propagation

During the test generation process of a binary network, an error signal is generated at a given point (line) in the circuit. Then, it is propagated to an observable output.

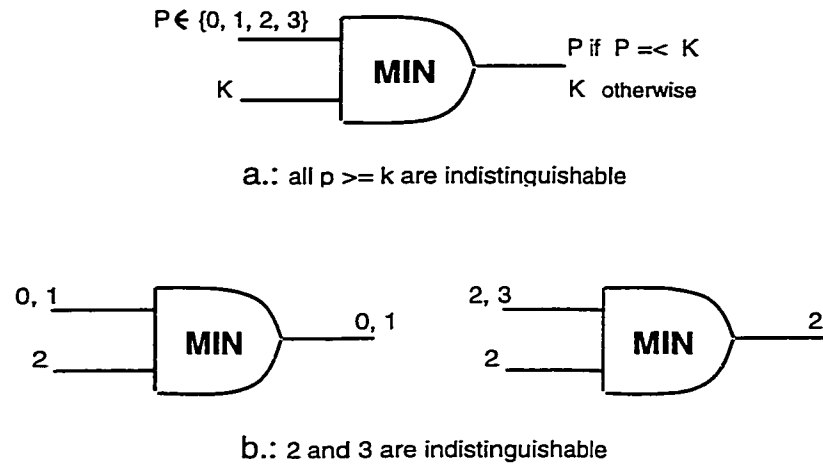


Figure 2.11: k-enabled gates.

During this propagation, assignments are made to some other circuit lines. These assignments have to be justified after a successful propagation of the error signal to an observable output is achieved. If both phases of error propagation and assignments justification are successful, a test is generated for the error signal (the fault). If, on the other hand, the assignments are not justified they are removed from the solution space and the process is repeated with new assignments. This procedure continues until a test is found or all possible assignments are tried. Usually, a time limit is set and when it is exceeded, the procedure stops even if there are untried assignments left. If the procedure stops and no test was found, the fault is considered un-testable. Different algorithms, e. g. D [30], PODEM [13] & FAN [11], try to guide the assignments through the solution space in such a way that minimizes the dead iterations (unjustifiable assignments) and avoids wrong assignments as early as possible. The degree of success in doing so determines the efficiency of the algorithm

and its execution time.

For MVL, similar procedure applies. The fact that lines can be assigned to more than two logic values introduces an added complexity. K-enabled gates also add to this complexity and if not treated properly, wrong assignments can be made during error propagation and a lot of time can be wasted.

Consider, for example, the 4-valued *min* gate shown in Figure 2.12. The gate is fully enabled since its fixed input (line b) is assigned to $r - 1 = 3$. Suppose that a value $p \in \{0, 1, 2, 3\}$ is assigned to the free line (line a). This value of p will test for all single stuck-at faults except stuck-at- p (s-a- p). To test for s-a- p , any value other than p can be used. For example, 0 will test for s-a-1, s-a-2, s-a-3. Any other value (1, 2, or 3) can test for s-a-0. As a result, a combination of any two values will test for all single stuck-at faults on the free line (line a). Vectors including 2 and 3 can be selected as shown in the figure.

Now, consider the partially enabled *min* gate shown in Figure 2.13. In this case, 0 is the only assignment that can test for s-a-1, s-a-2, s-a-3 giving 0/1, 0/1 and 0/1 output respectively. Any other assignment will only test for s-a-0 (1, 2, or 3 will give 1/0). So, unlike the previous case, the combination 2 & 3 can not be used. Any valid combination must include the input value 0.

This illustrates that k-enabled gates and paths passing through them should be

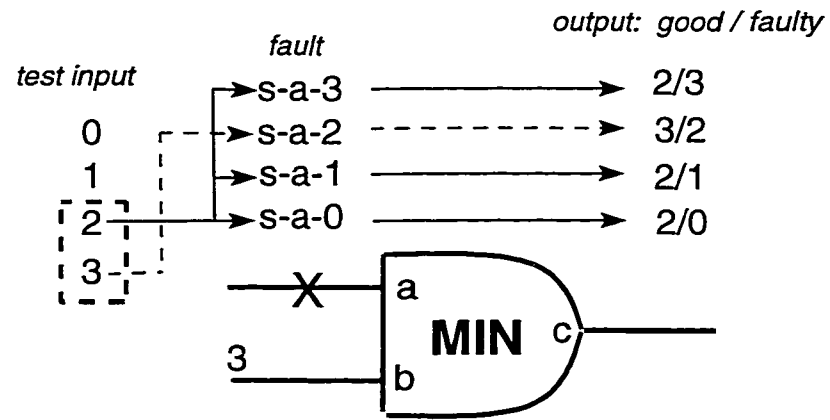


Figure 2.12: Fully enabled gate.

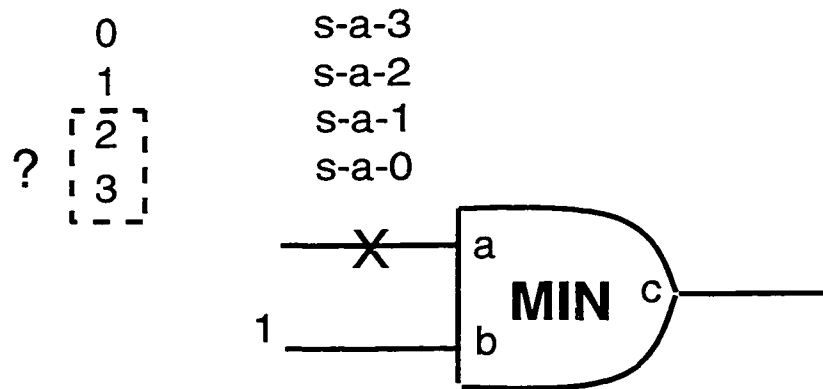


Figure 2.13: 1-enabled gate.

treated properly in any test generation algorithm. A significant time can be wasted during test generation if the k -enabled gates concept is not taken into account by algorithms generating tests for MVL circuits. The occurrence of these k -enabled gates is network dependent (related to the values of constants, k 's, found in the network). Also, every MVL network is likely to have k values (constants) since any set of MVL operators must include a decision operator that limits its output to two values for all input combinations. These values become the k 's for gates connected to that output. For example, the literal operator $2[\{x\}^2]$ will give either 0 or 2 on the output. Then if 2, for example, was connected to a *min* gate, it will limit its sensitivity. Values ≥ 2 on the other input of the *min* gate will not produce distinguishable output changes, as shown in Figure 2.11(b).

Chapter 3

Literature Review

Previous work showed that it is more involved to test MVL circuits as compared to their binary counterparts since the test generation algorithms have to deal with the new phenomena introduced by MVL operators. Spillman and Su [39] modified the D-algorithm to test MVL circuits. They start by identifying k-enabled gates and paths passing through them for the whole circuit. Then special D's are propagated through these paths. For the other paths, D propagation is the same as in the binary case. The D-algorithm is modified to propagate D-chains along three separate types of paths:

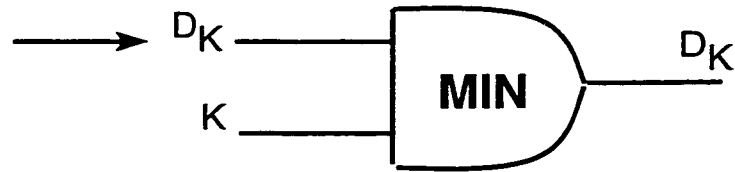
1. paths along min gates where the other input of the min exists on a k-path (k-enabled min). A simplified example is shown in Figure 3.1(a). The D's along

such paths are subscripted with a k . Stuck-at-p faults are indistinguishable for $p \geq k$.

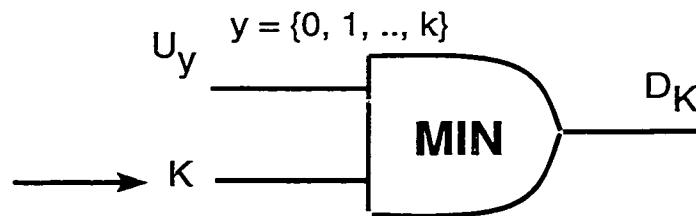
2. D-chains along k-paths where a k value exists on any point of the path. This is illustrated in Figure 3.1(b). The U_y notation is introduced to show the limited set of values that a line can have. The set $y=\{0, 1, \dots, k\}$ is the set of values that a line can not assume.
3. D-chains along all other remaining paths (normal D-chains).

The modified D-algorithm, first, constructs all tests for distinguishable faults along the k-paths of the circuit. Then, it constructs the tests for distinguishable faults along D_k chains. Finally, it constructs tests for faults along the remaining D-chains. Further, the introduced Δ product algorithm reduces the number of tests by utilizing the U notation.

Tabakow [42] introduced a generalized D-algebra. The purpose was to introduce an algebraic system usable in the process of fault oriented test generation for combinational MVL circuits. This method simplifies path sensitization as compared to the previous method of Spillman and Su. In this system, D's are represented as: d_{ij} where $i, j \in \{0, 1, \dots, r - 1\}$ and r is the radix of the MVL system. The binary D, for example, is represented as d_{01} . For a ternary algebra, d_{02} means 0 in the fault-free circuit and 2 in the faulty circuit. Constants like 1 and 2 are represented by d_{11} and



a.: propagating along k-enabled gates.



b.: propagating along k-paths.

Figure 3.1: Different D-chains propagation.

d_{22} , respectively.

The error signal and the generated d_{ij} 's are propagated by means of intersection rules defined for the system. These rules will only propagate the correct d values through any k-enabled gate directly without the need for a second pass like in the Spillman and Su method. Some of these rules are introduced below with some examples assuming ternary logic ($r=3$).

$$\text{MAX}(i, j) = i + j = i \text{ if } i \geq j \text{ and } j \text{ otherwise.}$$

$$\text{MAX}(d_{i_1 j_1}, d_{i_2 j_2}) = d_{i_1 j_1} + d_{i_2 j_2} = d_{(i_1+i_2)(j_1+j_2)}.$$

$$\text{MIN}(i, j) = i \cdot j = i \text{ if } i \leq j \text{ and } j \text{ otherwise.}$$

$$\text{MIN}(d_{i_1 j_1}, d_{i_2 j_2}) = d_{i_1 j_1} \bullet d_{i_2 j_2} = d_{(i_1 \bullet i_2)(j_1 \bullet j_2)}.$$

As mentioned earlier, if a 2-input min gate, for example, is k enabled, then it is sensitive only to inputs in $\{0, 1, \dots, k-1\}$ where $k \in \{0, 1, \dots, r-1\}$. Accordingly, the d_{ij} values that can propagate through should satisfy:

$\text{MIN}(k, d_{ij}) \in D_{\neq} \leftrightarrow k \in M - \{0, \dots, \min(i, j)\}$ where $M = \{0, 1, \dots, r-1\}$ and D_{\neq} is the set of all $d_{ij} / i \neq j$ (a d value that distinguishes the fault-free circuit from the faulty circuit). Similarly, the rule for the MAX operation:

$\text{MAX}(k, d_{ij}) \in D_{\neq} \leftrightarrow k \in M - \{\max(i, j), \dots, r-1\}$. Similar rules are defined for the rest of operations, formalization of propagation rules and definitions are presented to complete the algebraic system. Also, the system is generalized to deal with multiple stuck-at faults.

Also, Vlad Shmerko et. al. [37, 36] modified the D algorithm. However, they used modern results of Logic Differential Calculus and also sensitizing path method to build generalized D-algebra based on the Direct Logic Derivative (DLD) notation.

In parallel, other works have used algebraic methods to find MVL tests. Ajab Noor and Abd-El-Barr [1], for example, used Boolean Difference-like method in testing for stuck-at faults in MVL circuits. Tapia and Guima [43] introduced new differential operators for logic functions in a multi-valued algebra. These operators were used in algorithm to find complete tests for stuck-at type faults in MVL circuits.

They showed that the boolean differential operators are special cases of the new MVL operators. Whitney and Muzio [47] generalized the decisive difference method to MVL and used it to define functional transformations as partial differences. Boolean difference is a special case of the partial differences applied.

Other works used some new concepts in finding MVL tests. Dubrova et. al. [8, 9] used the full sensitivity concept, found in MVL, and formalized it to generate expressions to be used in test generation. Basically a function is fully sensitive to some line l if each transition on this line from one logic level to another causes a change in the output logic value. This concept is formalized and used to calculate test vectors for large functions. For example, if a line l in an MVL n -variable function is to be tested for a stuck-at- p fault, it is cut and considered as pseudo input x_l . Then, the line value is expressed in terms of the function inputs $l(x_1, x_2, \dots, x_n)$. The function is expressed in terms of its inputs and the line l : $f_l(x_1, \dots, x_n, x_l)$. Then, the formalized procedures are used to find some set of values (a_1, \dots, a_n) such that $l(a_1, \dots, a_n) \neq p$ and some $k \neq p$ such that $f_l(a_1, \dots, a_n, k) \neq f_l(a_1, \dots, a_n, p)$. A solution for two different values of k will result in tests for all stuck-at faults at line l .

In the work of Wang et. al. [46], a complete test set (CTS) was defined and derived for MVL Min/Max networks. The CTS is capable of detecting any single and multiple stuck-at faults of MVL Min/Max networks. Once this CTS is found

for a given function, it will be valid regardless of the implementation. It is similar to the binary Universal Test Set (UTS). The method of generating the CTS starts by defining the literal truth table for the given function. First, the function is described in Sum of Products of literals. There is one column, in the literal truth table, for each literal in the function expression and one row for each possible function value (listed in the row, the literal values that produced this function value). Then, by applying enumeration rules and algorithms, the CTS is derived for the function. This method is reported to be more efficient and time and memory saving than enumerating the truth table of the function.

There has been some work reported in the literature on testing Sequential Multi-Valued Logic (SMVL) circuits. Drechsler et. al. [6] presented a fault simulator for Sequential Multi-Valued Logic Networks (SMVLN). This fault simulator was an extension of the combinational circuits fault simulator presented in [7]. The simulator algorithm receives r (the radix of the r -valued MVL), an input sequence, and a description of the circuit as input and returns the fault coverage achieved as output. In addition, the set of undetected faults is returned. The algorithm works in two main steps. First, the list of all possible faults is generated and then simplified by finding the set of equivalent faults. In the second step, the detectability of these faults is determined by an event-driven single fault propagation procedure. In this procedure, the circuit is first evaluated under the input sequence and the value of

each line is determined. Faults are then injected into the circuit one by one and only events produced by an inserted fault are propagated to the primary outputs. If, during this propagation, any primary output changes its value, the fault is marked detected and removed from the list. To illustrate the event-driven approach, consider Figure 3.2. Suppose that $r = 3$, the input vector is $[a = 2, b = 2, c = 1]$ and node b is stuck-at 1. Both *min* gates have to be re-evaluated which results in a new event at node u . Accordingly, the *max* gate have to be re-evaluated and the fault is marked detectable since it produces a change at the primary output w . To handle SMVLNs, the fault simulator was extended by replacing each memory element by a Secondary Input (SI) and a Secondary Output (SO) [6]. After this replacement, the SMVLN is converted to MVLN where the value of any SI at time t is defined by the value of the corresponding SO at time $t-1$.

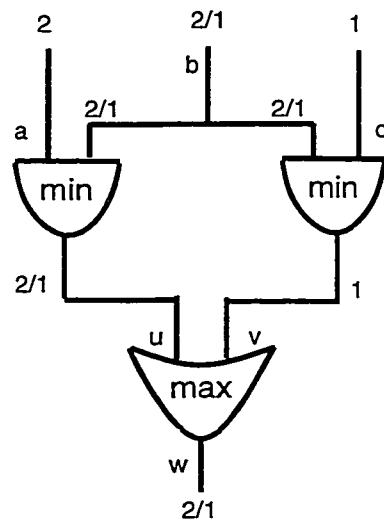


Figure 3.2: Event driven fault simulation.

Keim et. al. [23] presented an Automatic Test Pattern Generation (ATPG) tool for SMVLN based on Genetic Algorithms (GA) and reported experimental results for large circuits, with up to some thousand gates. The fault models considered are the stuck-at and skew fault models. There are two types of skew faults. One raises the logical value of an edge in the circuit by one, and the other lowers the logical value of that edge by one. The basic concept in GA is: Two parent elements (genes) are mixed (GA operator) to produce two new generation elements (genes) each of which has a combination of properties from the two parents. In addition to the standard GA operators used, a problem specific operator (**Free Vertical Crossover**) is defined. The test sequence set is represented by a two-dimensional matrix. The x-dimension represents the number of inputs and the y-dimension represents the number of test patterns (see Figure 3.3 [23]). Free Vertical Crossover is defined as: construct two new elements c_1 and c_2 from two parents p_1 and p_2 . Determine for each test vector t a cut position. Divide each test vector t of p_1 and p_2 in two parts at its cut position. The first (second) part of each test vector c_1 (c_2) is taken from p_1 and the second (first) part is taken from p_2 . Figure 3.3 illustrates this operator (the black areas are filled with random patterns). The main idea of the ATPG tool introduced in this work makes use of GA in two phases. The first phase optimizes a given test sequence set for the length and fault coverage. The algorithm in this phase runs until no improvement found for 100 consecutive generations. The second phase increases the size of the set to improve the fault coverage of uncovered

faults in phase one. The stop criterion for the second phase is set after 50 consecutive generations with no improvement.

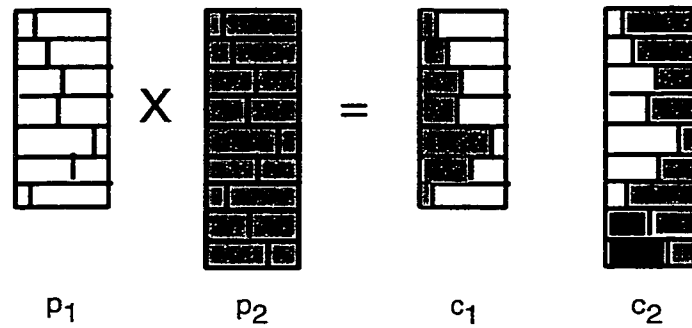


Figure 3.3: Example for *free vertical crossover* genetic operator.

Other category of research used MVL circuits and concepts to test binary circuits. Mou Hu [18], for example, discussed Design for Testability (DFT) rules to design binary systems based on ternary logic. Two values of the ternary logic were used for normal binary operation. The third value was used only for testing purposes. The third value will appear only in the test vectors and the function's normal response to these vectors. If the function undergoes a fault, its response to the test vector will be one of the binary values indicating the presence of a fault.

Chapter 4

Physical Defect Modeling and Simulation

4.1 Physical Defects

On the production lines of ICs, physical defects interfere with the desired circuit design and result in faulty chips. The occurrence of such defects depends on the circuit layout, fabrication process, device parameters and many other technology related factors.

There have been tremendous efforts to understand and characterize physical de-

fects in ICs and their effect on the circuit behavior. Many studies were conducted on faulty chips taken from actual production lines [12, 2, 40, 26, 29, 5]. The actual reasons (physical & chemical) behind such defects are not well known. However, statistical data collected from experiments and actual production lines is available in the literature. This becomes an important input to realistic fault modeling studies producing fault models closer to the actual physical defects and guiding test generation to more accurate decisions.

In general, defects can be classified into two groups:

1. Shifts in electrical parameters (voltage, threshold, etc.): This can grossly affect the operation of the chip and can be easily detected by parameters measurements.
2. Topological layer deformation: This can be classified further into two types:
 - (a) Global: such as scratches across the whole chip, photolithography misalignment, line registration errors, too thick gate oxide, too thin polysilicon, etc. These defects are easily detected by almost any test pattern. This is because they tend to have global effects on the chip operation and functionality.
 - (b) Local (spot): such as gate oxide pinhole, dust particles on chip or mask. These defects result in complete, or nearly complete, opens or shorts in

some signal lines and affect small (local) portions of the chip producing the hardest faults to detect [4, 5, 19].

4.1.1 Physical Defects & Faults

Defects at the lowest level, the layout level, manifest themselves in a variety of forms at higher levels, electrical or logical level. The description of these effects at high levels is known as *fault modeling*. Fault models should compromise two conflicting features: the needs for accurate modeling of physical defects and easy-to-deal-with models that will not complicate the test generation process [19].

For long time, test pattern generation algorithms relied on the assumption that defects can be modeled at the gate level representation as lines stuck-at 0 or 1. However, as VLSI device sizes decrease and with the emergence of newer technologies, e.g. CMOS and BiCMOS, the adequacy of such assumption becomes questionable. A number of studies showed that the stuck-at model is not adequate to model failure mechanisms found in practice. For CMOS technology, most of the defects result in undesirable bridges (shorts), including gate oxide shorts, breaks (opens) and circuit parameter shifts. These can not be accurately modeled using the stuck-at fault model [12, 2].

4.1.2 Spot Defects

The probability that a spot defect will cause a fault depends on the spot size, spot location, and circuit layout topology. For example, the probability that a spot will short two wires running in parallel depends on the wires's length, spot size (compared to the separation distance), and the location of the spot [5]. If the line width is w , separation distance of the two lines is d , and the spot is represented by a circle of diameter x , then the probability of shorting two lines, or more, can be estimated once these dimensions are known (See Figure 4.1) [19].

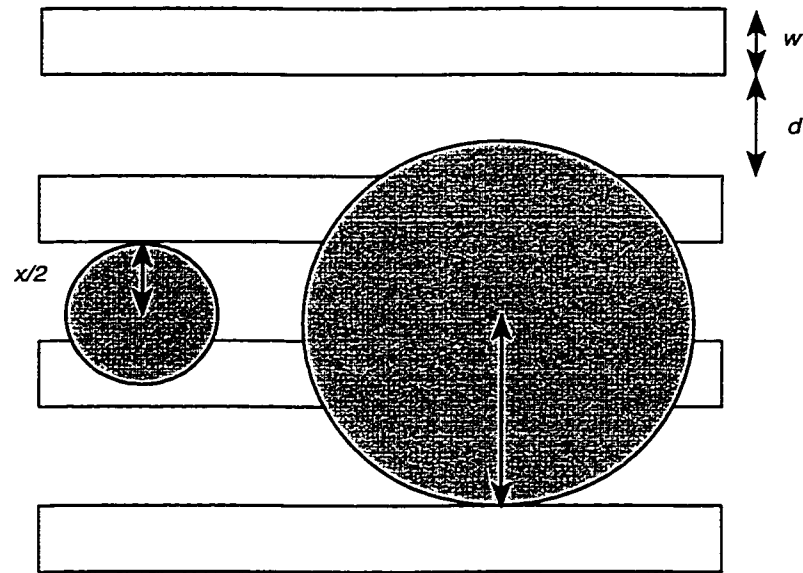


Figure 4.1: Spot defects overlapping two or more wires.

The *critical area* is defined as the area where the spot center is expected to occur [4]. For example, the overlapping area of two conducting lines of adjacent layers is critical. Stapper [40] found mathematical models to represent these defects. The

models related the critical area to the defect size and location. Defect size and location distribution functions were obtained from statistical data gathered from different production lines and sites. The critical area can be calculated from the process masks. For example, a mask-dependent defect size distribution function D_m was developed as:

$$D_m(x) = \begin{cases} \overline{D_m} \frac{x}{x_0^2} & \text{for } 0 \leq x \leq x_0 \\ \overline{D_m} \frac{x_0^2}{x^3} & \text{for } x_0 \leq x \leq \infty \end{cases}$$

where x is the spot circle diameter, x_0 is a technology dependent constant, and $\overline{D_m}$ is the average defect density for layer m . The average number of faults $\lambda_{photo,m}$ (produced at a layer creation process defined by mask m and defect sensitive area A) can be calculated as:

$$\lambda_{photo,m} = \int A(m).D_m(x)dx$$

These models were then used to estimate the fault occurrence and yield of different circuits. After studying the effects of spot defects on the circuit behavior, more realistic types of faults can be recognized which leads to a more accurate tests. Defects that create similar electrical behavior are grouped to produce fault lists. Typical failures found in CMOS technology include: bridging faults (most found), broken wires, missing contacts, extra contacts and newly created devices (low percentage). Such failures result in faults that can not be described as stuck-at faults [19, 10, 2, 5]. This shows the need for characterizing the fault types occurring in a specific technology and not relying only on the stuck-at fault model which in some cases, like

SRAM, represents only 50% of the actual faults [5].

4.2 Fault Characterization

In general, to find the effects of defects on the circuit behavior (**fault characterization**), a defect model is proposed and inserted in the circuit. Then, simulation is performed to find the circuit behavior under that defect. This process can be done at two different levels, namely at the device level [34, 44, 10, 48, 15, 17, 24, 25, 27, 31, 32, 41] or at the layout level [4, 5, 35, 40, 26, 19].

4.2.1 Device Level Fault Characterization

The gate level representation of a circuit does not relate directly to the devices locations and layout. There may be lines and connections at the device level representation which do not exist or have no meaning in the gate level representation. For example, the gate-level symbol of a NAND gate does not show the lines connecting its internal transistors. Hence the device level representation is believed to give more accurate representation of the actual circuit elements.

Based on the studies mentioned in the previous section and other studies on the effect of transistor gate-oxide shorts [34, 44], a general procedure for device level

fault characterization is widely accepted in the literature. Given the circuit level representation, defects are inserted one at a time. Then, using a standard circuit simulator (e.g. SPICE), the behavior of the circuit is evaluated in the presence of the defect to recognize fault types and groups.

At the device level, the possible faults considered are shorts between terminal nodes (gate-drain, gate-source and source-drain) and opens between these terminals and other circuit nodes connected to them.

Device level fault characterization found in the literature can be classified into two main groups. The first one represents shorts as zero resistance connections. Favalli et. al. [10] simulated faults and compared the analog voltages at relevant nodes with the logic threshold voltage V_{it} to find out if the fault generates faulty logical behavior. This gives detectability tables, for each type of fault for each gate, which are used by the simulator to find whether or not the fault is detectable.

The work of Zaghloul et. al. [48] also falls into this group. They proposed a graphical technique in which the transfer characteristics of the circuit's basic elements are graphically intersected to produce the resultant characteristics of a combination of elements at a higher level. This process starts at the faulty part and proceeds to other affected parts up to the primary outputs. This gave more accurate results in evaluating the complex analog behavior of some faults, such as gate-oxide shorts.

The second group takes into account the variable short resistance actually found in practice. Hao and McCluskey [15] showed that for typical CMOS circuits this resistance value, in some cases, is critical and circuit behavior varies largely for different values of this resistance. Some shorts can produce normal output, delayed output (delay fault) or logical fault depending on the resistance value of the short.

Similar studies were conducted on BiCMOS technology [17, 24, 25, 27, 31, 32, 41]. These studies showed different behaviors of circuits that can not be detected by tests generated for stuck-at faults. Ma and McCluskey [25], for example, found new types of faults in BiCMOS that do not exist in other technologies. In some cases the faulty circuit produced the correct output for some time and then transferred to a faulty value. In other cases the output was oscillating.

This shows that technology and circuit design and topology largely affect the type of faults expected to occur during fabrication, and highlights the needs for techniques to test for these faults.

For this class of fault characterization, it is common to represent the defects at the transistor level as resistive shorts between terminals and opens between each terminal and circuit nodes connected to it, see Figure 4.2.

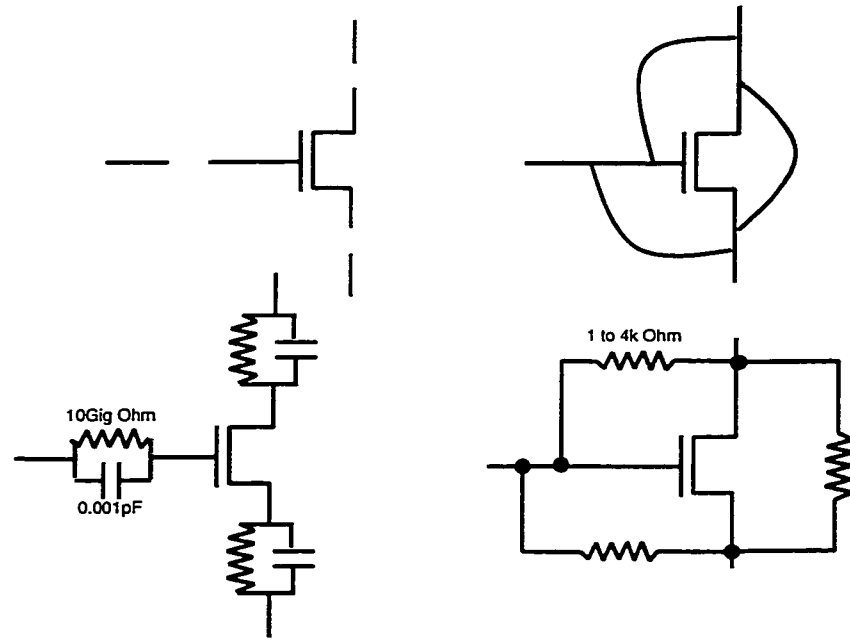


Figure 4.2: Open and short faults and their models.

4.2.2 Layout Level Fault Characterization

Although the device level representation is more accurate than the gate level in representing the actual circuit, the layout level deals with the actual circuit representation and highlights the technology and circuit topology dependent features. Generally, layout fault characterization starts by inserting defects at the layout of the circuit. Then, the resultant layout is submitted to an extractor to extract the circuit representation of the modified layout. This circuit representation can then be simulated, using a circuit simulator, to compare the behavior of the new circuit, with the inserted defects, to that of the original one. The results of the previous step are gathered and classified to produce fault lists that are directly related to the

actual layout. A typical example is the work of Chess et. al. [4] and Dekker et. al. [5].

Starting from the layout, defects are inserted into the various masks and the resultant layout is simulated to extract the faults. Inductive Fault Analysis (IFA) [35] is a systematic procedure in which defects are inserted in the layout and the resultant faults are extracted. Spot defects are the most important to consider. Spots are represented by circles of various radii. At any mask, a circle can be opaque or transparent resulting in a presence or absence of the corresponding material(s) defined by the mask. The main input to this stage contains data on probability functions for spot sizes and distributions. This reduces the huge size of the work space that considers all possibilities. Actual formulas and explanations for the occurrence of defects and probability distribution functions are not known. However, these relations and formulas are estimated from practical experiments and data collected from actual production lines. Examples are found in [40, 26]

After the process of spot insertion in the layout, a process of elimination starts. The spot size and location determine whether or not the spot is potentially going to produce faults at higher levels. For example, a spot falling between two parallel lines will not short them unless its radius is larger than the separation distance. Also, it should be located properly, i.e., contacting both lines.

Propagating to a higher level, the resultant defects from the previous stage are evaluated for their electrical significance. A process of elimination is repeated here too. A defect which shorts two areas is critical at the layout level but it may be insignificant at the electrical level if the short, for example, connects two nodes which are designed to be at the same potential.

The previous stage produces technology dependent primitive defect types which can be produced once the technology is defined and then used in fault extraction of any given circuit. The technology related primitive faults are then examined, simulated and grouped giving circuit faults that can be represented at the device level and simulated. The resultant fault lists represent the technology and layout related defects. Further more, these fault lists are ranked according to the probabilities of defect distributions that caused them. It is possible that a number of defects lead to the same fault. So, the larger the number of defects that caused the fault, the more likely it will be found at the end of a production line. Taking this into account can lead to more efficient testing. Details of IFA are found in [35].

Another class of layout dependent fault analysis is found in [19]. Their work is based on the same principles of IFA. However, instead of the complicated and time consuming simulations, this work developed a set of mathematical formulas that describe fault probabilities and then extract faults from the circuit layout efficiently. The formulas and relations are based on the work of Stapper [40].

4.2.3 Comparison

Layout level fault characterization is more accurate in representing the actual physical defects. However it is related to the circuit layout and the process masks. If the circuit layout or the process is changed, the characterization has to be repeated. On the other hand, device level characterization is independent of the layout or the process. Also it is easier to-deal-with and less time consuming. In this thesis, device level characterization will be used.

Chapter 5

Fault Characterization of the

MVL Set

The aim of the research work in this thesis is to characterize, at the device level, faults in the set of MVL circuits presented in chapter 2. Defects will be inserted into each circuit in the form of shorts and opens. Simulations, of the resultant circuits, will then be conducted. The circuits behavior in the presence of the defects will be classified into fault categories to give statistics on the types of faults expected to be found in these circuits.

5.1 The Approach

The functionally complete set of MVL operators, introduced earlier, will be used. It is implemented using the standard binary CMOS technology. This enables us to use the same techniques used for binary CMOS fault characterization. In this work, fault characterization will be conducted at the device level. The methodology used for fault characterization is explained below:

1. Defects are inserted on transistor terminals (3 shorts and 3 opens) and considered one at a time. This will be performed for each operator circuit.
2. SPICE will be used to determine the output of the circuits in the presence of each inserted defect.
3. Simulation results will be analyzed to provide statistics on faults and their occurrences.
4. Recommendations for MVL testability will be made based on the characterization results.

The resistance of the inserted short faults will be varied. In the literature, typical resistance values considered for CMOS technology vary from several Ohms up to 4k Ohms. Shorts may occur with higher resistances but with low percentage and very

high resistance value that will not affect the circuit behavior [38, 16, 14, 15, 25, 41]. Opens are modeled as 10 Gig. Ohms resistor in parallel with a 0.001pF capacitor between the transistor terminal and the node it is normally connected to. Figure 5.1 shows the short and open faults and their models.

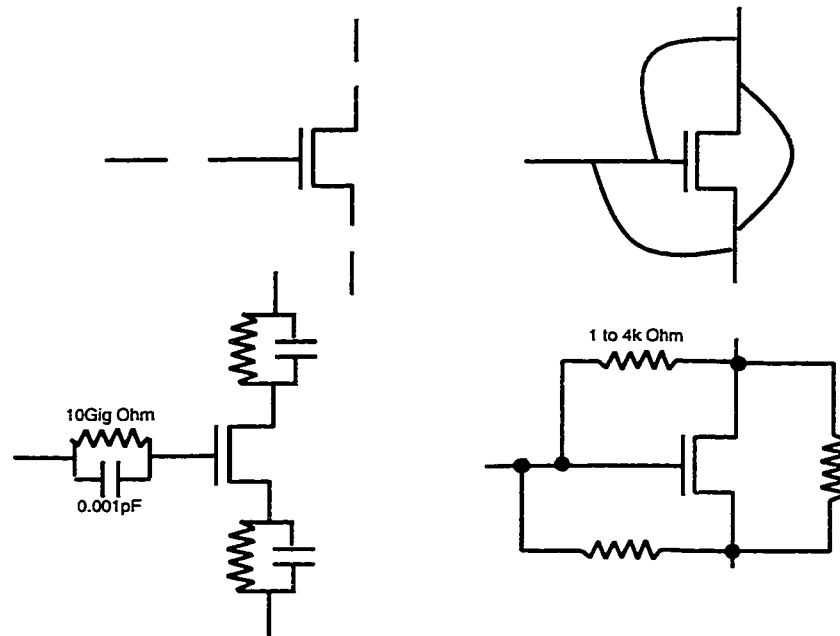


Figure 5.1: Open and short faults and their models.

For the following sections, the transistors in each circuit will be numbered arbitrarily starting with Current Mode CMOS (CMCL) transistors followed by Voltage Mode CMOS (VMCL) transistors. An open at the gate of transistor number 1, for example, will be referred to as O1g. O1d and O1s will refer to opens at transistor 1 drain and source, respectively. S1gs refers to the inserted short between the gate of transistor 1 and its source. Each circuit will be presented in a separate section. Simple description of the circuit operation will be given followed by some examples

of how this normal operation is affected under the presence of a fault. Each section will end with a table showing the fault categories found after the exhaustive simulations along with their statistics.

The following terms and abbreviations will be used:

- **SA k** means the output stuck-at k under the simulated fault.
- **aSA0** faults in the literals caused the value of a in $k^a[x]^b$ to be as if it was 0. For example, if the original circuit is $k^1[x]^1$, the resultant circuit will be $k^0[x]^1$.
- **bSA3** faults convert the circuit to $k^a[x]^3$. This has no effect if b in the original circuit was 3.
- **Sequential** faults: In the presence of a sequential fault, the output of the circuit depends on both the current input and the previous inputs of the circuit. For example, let x_i be the current input applied to a circuit having a sequential fault. Two different outputs may be observed, for the same x_i , depending on the value of the previous input x_{i-1} . Assume the two input sequences $x_{i-1} \rightarrow x_i = 0 \rightarrow 1$ and $3 \rightarrow 1$. The output, when x_i is applied, can be 3 for the first sequence and 0 for the second sequence. So, for the same x_i , two different output values are observed, depending on the previous input x_{i-1} .

- **Functional** faults change the circuit's behavior into a different function from what it was designed for. The output does not show any stuck-at behavior but gives faulty results for some (all) inputs.
- **Para_k** fault refers to a parametric change in the value of k . This value is supposed to be the same for the whole input range in $[a, b]$. In the presence of this fault, k takes several current levels, depending on the input value, which may be read as different logic values. For example, if the circuit $3^1[x]^3$ undergoes this fault, the values of the output current may be different from 60uA (the correct value) for different inputs. For example it could be 15, 40, 60 uA for $x = 1, 2$ and 3 , respectively. This implies that even if the circuit shows the correct output for some input values in the $[a, b]$ range, it could be wrong for others. This implies that the output should be checked at every possible input value in the range.
- **3* faults:** the normal value of the 3 logic level is 60uA. 3* faults cause this value to be higher than 60uA. Typical examples found were in the range of 80 up to 300uA. Also when a fault category is marked with a '*', this indicates the existence of a 3* behavior in addition to the other behavior. Sequential*, for example, indicates that the fault is sequential and produces 3* output.

Test generation for these faults (in all considered circuits) will be considered in the next chapter.

5.2 Fault Characterization of the Literal

Figure 5.2 shows the literal implementation with the transistors numbered as suggested before. Like the other circuits implementations, except for the *min*, it is composed of two parts. The input/output part is implemented using CMCL transistors (transistors 1, 2, 3, 4, 5, 6 and 7). The second part determines the logic part and consists of VMCL transistors (transistors 8, 9, 10, 11, 12 and 13). Depending on the inputs, the logic part controls the status of the output. Transistors 8 and 9 implement the inverter, in the literal realization shown in Figure 2.6, and feeding one of the inputs of the NOR gate (transistors: 10, 11, 12, 13). Transistors 4 and 5 are current to voltage transistors controlling the logic (binary 0 or 1) at points A and B, respectively. Their sizes are adjusted such that the voltage at point A will be binary 1 if the input current x is $\geq a$ and zero otherwise. The voltage at point B will be binary 1 only if the input current $x > b$ and zero otherwise (a & b in $k^a[x]^b$). For example, for $3^1[x]^1$, if the input is 0, transistor 4 will pass the mirror current of x causing the voltage at node A to be 0. As a result, the NOR output (transistor 13's drain) will be 0 closing the output switch (transistor 7) and hence the output current passing through transistors 6 & 7 will be 0. When the value of the input is 1 (in the range of $[a,b]$), node A will be charged to binary logic 1 and the inverter output (transistor 9's drain) will be 0. The other NOR input at node B will also be zero resulting in binary 1 at the output. This will open the output

switch and the current will pass through transistor 6. The size of transistor 6 is adjusted to pass constant current, 60 μA in this case (logic 3), which represents the literal gate output. Figure 5.3 shows the output of the fault free $3^1[x]^1$ under the input sequence $x = 0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 0$. The current passing through node A and the resultant voltage are shown in Figure 5.4. Node B current and voltage are shown in Figure 5.5.

Defects (shorts and opens) will be inserted one at a time. The circuit will then be analysed to find out the resultant behavior in the presence of the fault. This step will be repeated exhaustively for all defects (3 shorts and 3 opens per single transistor) to find resultant fault categories and their statistics. Some examples are given below.

Example 1: When the short S1gs is applied to the literal circuit, it produced a stuck-at 0 (SA0) fault for all inputs. It affects the mirror operation resulting in almost 0A currents going out of transistors 2 and 3. As a result, Node A and B voltages will be 0V, independent of the corresponding transistor size, and this will produce a 0 at the NOR's output which closes the output current path. The circuit will show a SA0 for all inputs. Figure 5.6 shows the input and output currents in the presence of the short S1gs. Figures 5.7 and 5.8 show the faulty currents and voltages for nodes A and B, respectively. The same simulation is repeated for all possible configurations of the literal, namely $k^1[x]^1$, $k^1[x]^2$, $k^1[x]^3$, $k^2[x]^2$, $k^2[x]^3$, $k^3[x]^3$ for all

possible values of k (1, 2 and 3). The fault produced the same effect, SA0, for all combinations and thus will be classified in the SA0 fault category.

Example 2: When the open O9g is applied, the circuit showed a sequential behavior. For different previous inputs of the circuit, different outputs are observed for the same current input. For the literal $3^1[x]^2$, the fault free output is 3 when the input value of $x_i = 2$ is applied. The faulty circuit's output is 3 (matching the correct output) if the previous input was $x_{i-1} = 3$ and 0 if $x_{i-1} = 0$. So, depending on the previous input, the current input can result into two different outputs. Almost all gate opens resulted in sequential behavior.

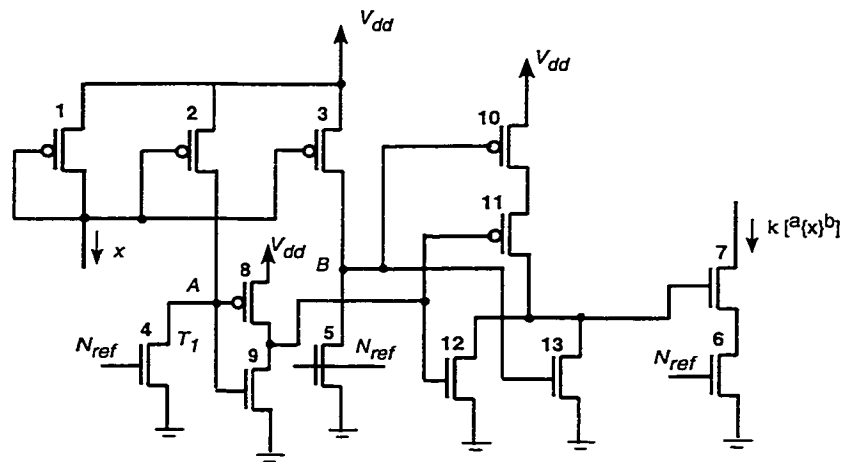
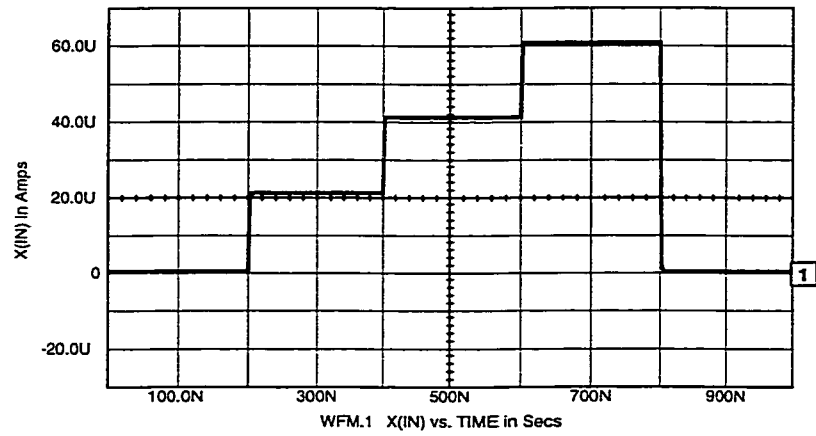
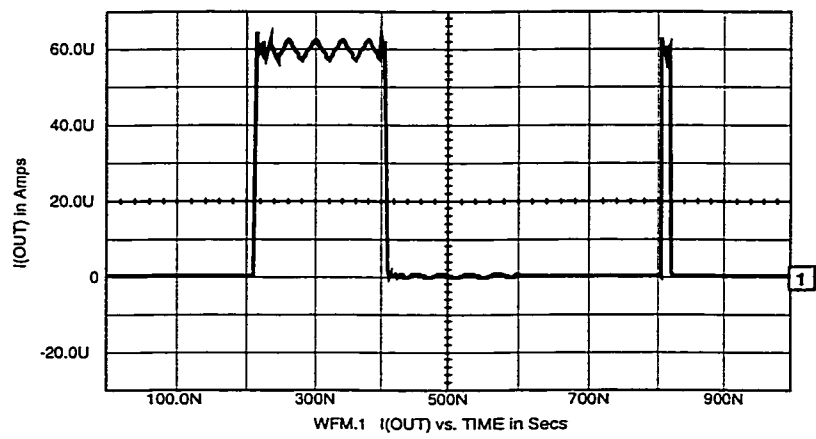


Figure 5.2: Literal transistor diagram.

Table 5.1 summarizes the simulation results obtained and the fault categories identified for the literal circuit. Four main fault categories were found: sequential,

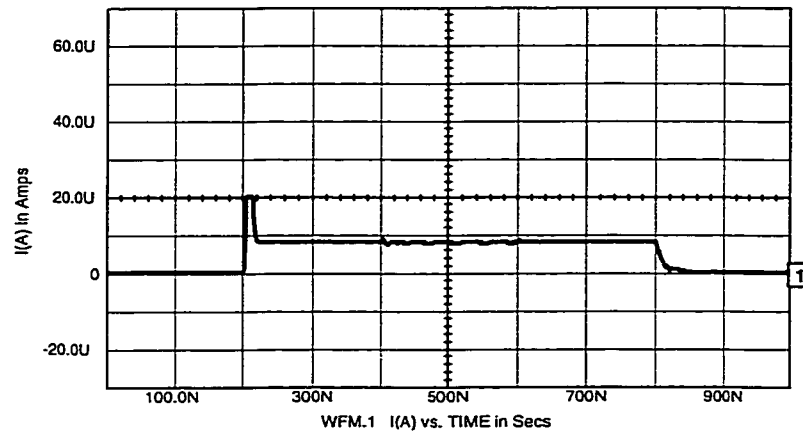


a. Input x

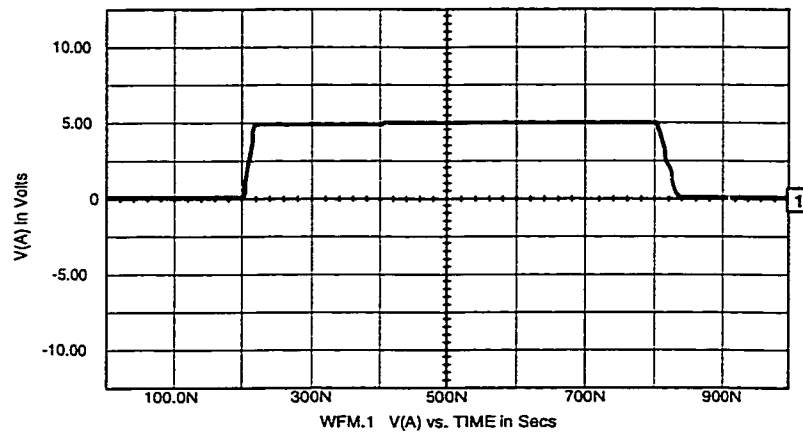


b. Fault free output

Figure 5.3: Response of the literal, $3^1[x]^1$.

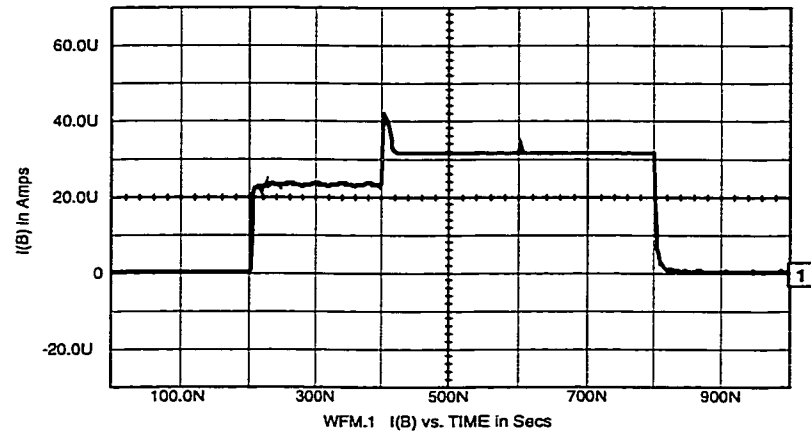


a. Current through node A

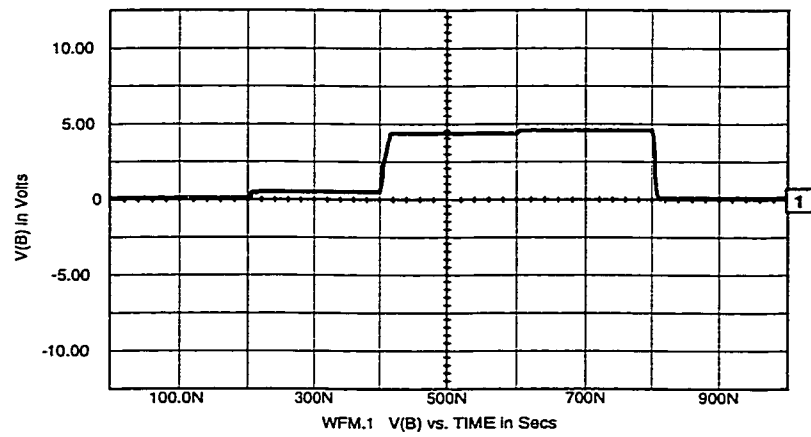


b. Node A voltage

Figure 5.4: Node A current and voltage.

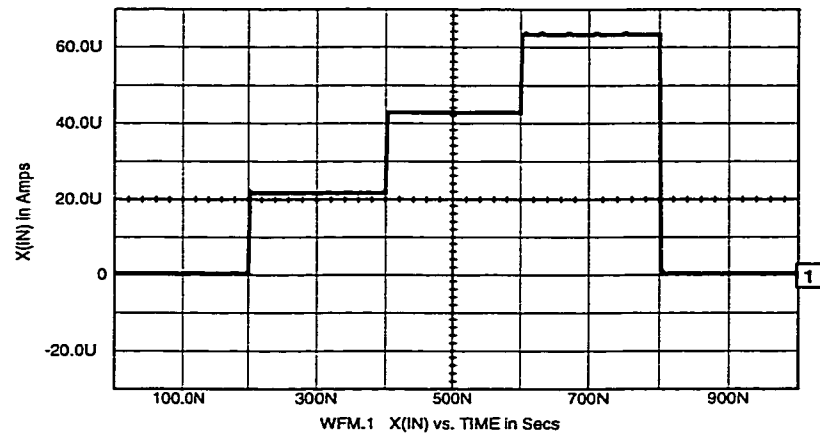


a. Current through node B

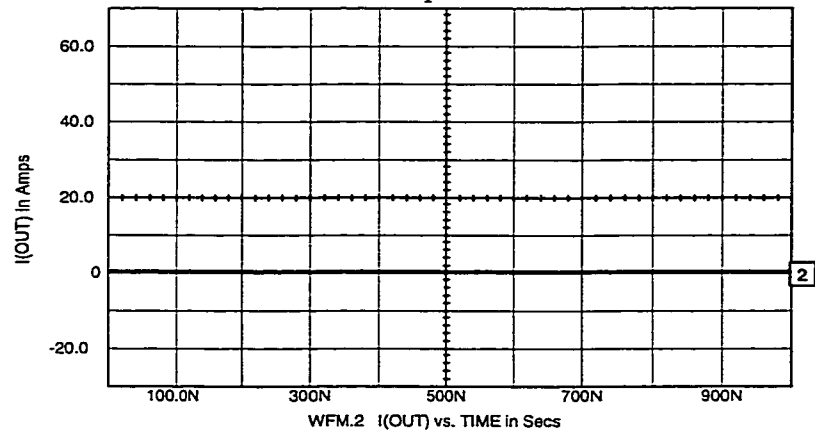


b. Node B voltage

Figure 5.5: Node B current and voltage.

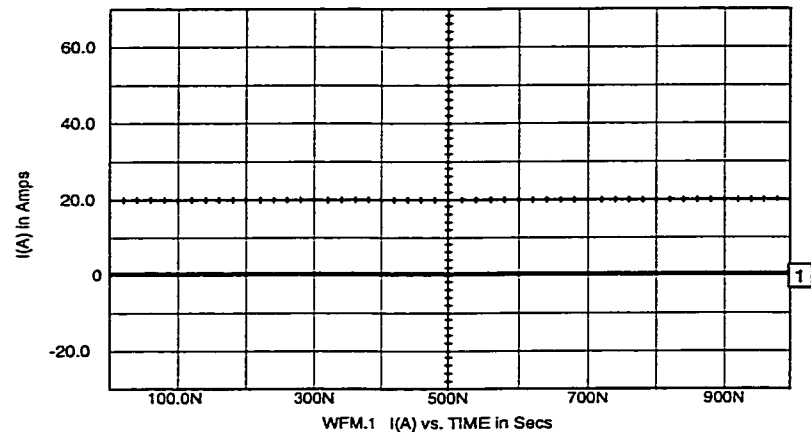


a. Input x

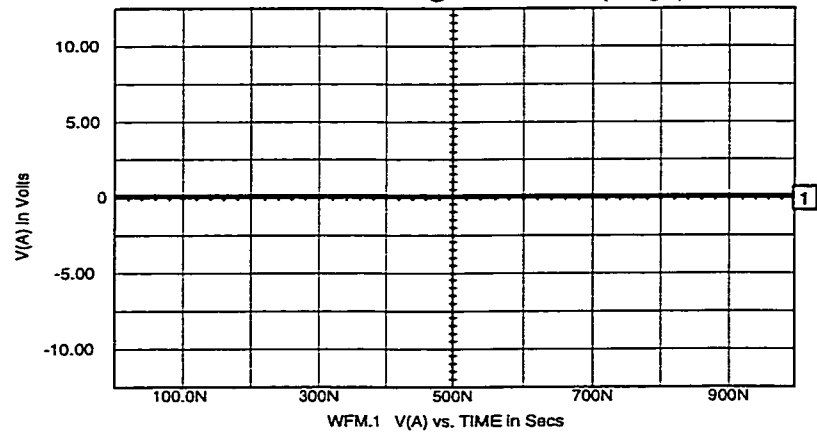


b. Faulty output, SA0 (Slgs)

Figure 5.6: Response of the circuit for literal $3^1[x]^1$ in the presence of Slgs.

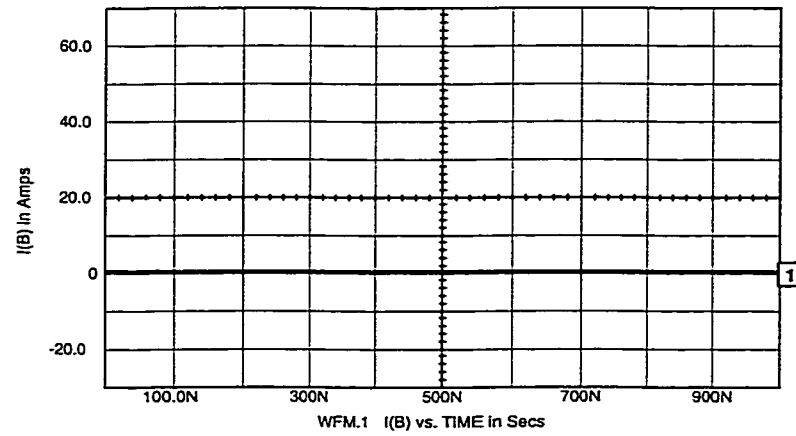


a. Current through node A (S1gs)

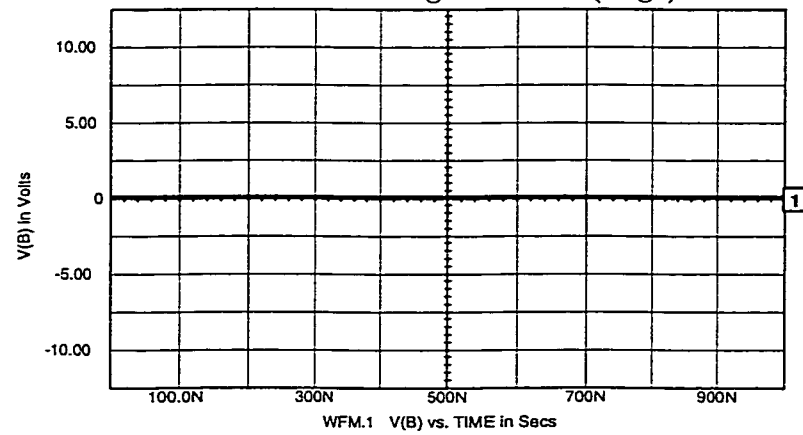


b. Node A voltage (S1gs)

Figure 5.7: Node A current and voltage in the presence of S1gs.



a. Current through node B (S1gs)



b. Node B voltage (S1gs)

Figure 5.8: Node B current and voltage in the presence of S1gs.

SA0, SAk, and functional. Faults resulting in 3* output occurred at low percentage, 2.6% (faults S6sd and O6g output). The highest percentage of faults, 40.3%, resulted in stuck-at 0 output (SA0). Sequential behavior was observed in 33.8% of the cases. Functional faults are listed and classified according to the resultant faulty functions into: aSA0, bSA3, bSA3 + para_k (both bSA3 and para_k observed) and complement (resulted exactly into the complement of the fault-free circuit). These totaled in 15 functional faults, 19.5%. The *others* fault category (in Table 5.1) includes Short S6gd, which resulted in oscillations (between 0 to 60 uA) at the output, and short S5gd, which resulted in irregular functional change that can not be classified in any of the above categories.

5.3 Fault Characterization of the Complement of Literal

Figure 5.9 shows the complement of literal implementation with the transistors numbered as suggested before. It is composed of two parts. The input/output part is implemented using CMCL transistors (transistors 1, 2, 3, 4, 5, 6 and 7). The second part determines the logic part and consists of VMCL transistors (transistors 8, 9, 10, 11, 12 and 13). Depending on the inputs, the logic part controls the status of the output. Transistors 8 and 9 implement the inverter, in the complement of literal

Fault Category	Faults	Count	Percentage
Sequential	O1g, O2g, O3g, O4g, O4s, O4d, O5g, O7g, O8g, O8s, O8d, O9g, O9s, O9d, O10g, O10s, O10d, O11g, O11s, O11d, O12g, O12s, O12d, O13g, O13s, O13d	26	33.8%
SA0	S1gs, S1sd, S2gs, S3gs, S3gd, S3sd, S4gs, S4gd, S4sd, S5gs, S6gs, S7gs, S8sd, S8gd, S9gs, S9gd, S10gs, S10gd, S11gs, S12sd, S13sd, O1s, O1d, O2s, O2d, O5s, O5d, O6s, O6d, O7s, O7d	31	40.3%
SAk	S7sd	1	1.3%
Functional	<i>aSA0:</i> S2sd, S2gd, S8gs, S9sd, S11sd, S12gs, S11gd, S12gd	8	19.5%
	<i>bSA3:</i> S5sd, O3s, O3d, S13gs	4	
	<i>bSA3 + para_k:</i> S13gd, S10sd	2	
	<i>complement:</i> S7gd	1	
	total 15		
3* faults	S6sd(kSA3*), O6g(sequential*)	2	2.6%
Others	S5gd, S6gd	2	2.6%
		total 77	

Table 5.1: Fault characterization of the literal circuit.

realization shown in Figure 2.7, and feeding one of the inputs of the NAND gate (transistors: 10, 11, 12, 13). Transistors 4 and 5 are current to voltage transistors controlling the logic (binary 0 or 1) at points A and B, respectively. Their sizes are adjusted such that the voltage at point A will be binary 1 if the input current x is $\geq a$ and zero otherwise. The voltage at point B will be binary 1 only if the input current $x > b$ and zero otherwise (a & b in $\overline{k^a[x]^b}$). For example, for $\overline{3^1[x]^1}$, if the input is 0, transistor 4 will pass the mirror current of x causing the voltage at node A to be 0. As a result, the NAND output (transistor 13's drain) will be charged to binary 1 through transistor 11 opening the output switch (transistor 7) and hence the output current passing through transistors 6 & 7 will be $k = 3$. When the value of the input is 1 (in the range of $[a,b]$), node A will be charged to binary logic 1 opening transistor 12 and closing transistor 11. Transistor 5 will still pass the mirror current of x resulting in 0 binary voltage at node B. This will drive the inverter output (transistor 9's drain) to binary 1 which is connected to the other NAND input. Since both inputs of the NAND are 1, its output will be binary 0 closing transistor 7 and the complement of literal output will be 0 current (logic 0).

Defects (shorts and opens) will be inserted one at a time. The circuit will then be analysed to find out the resultant behavior in the presence of the fault. This step will be repeated exhaustively for all defects (3 shorts and 3 opens per single transistor) to find resultant fault categories and their statistics. Some examples are

given below.

Example 1: The short S1gs which resulted in a SA0 for the literal, resulted in a SAK fault for the complement of literal circuit. Figure 5.10 shows the fault-free and faulty outputs of $\overline{2^2[x]^3}$ for S1gs (for the same input sequence $x = 0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 0$ used previously).

Example 2: From the sequential fault category, the open at transistor 10's drain (O10d) is illustrated below. Figure 5.11 presents the input and the fault free response of $\overline{3^2[x]^2}$. The faulty sequential behavior is shown in Figure 5.12 along with the input (repeated for clarity). Two different output values (intervals: 0-200 ns and 800 - 1000 ns) are found for the same input value $x=3$.

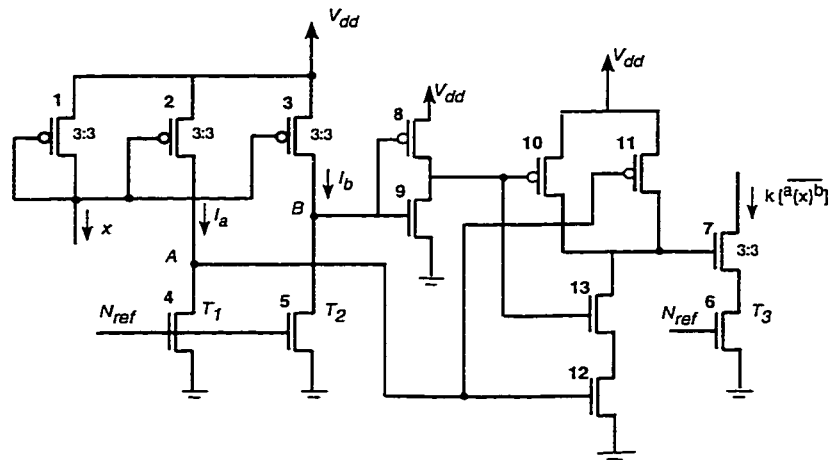
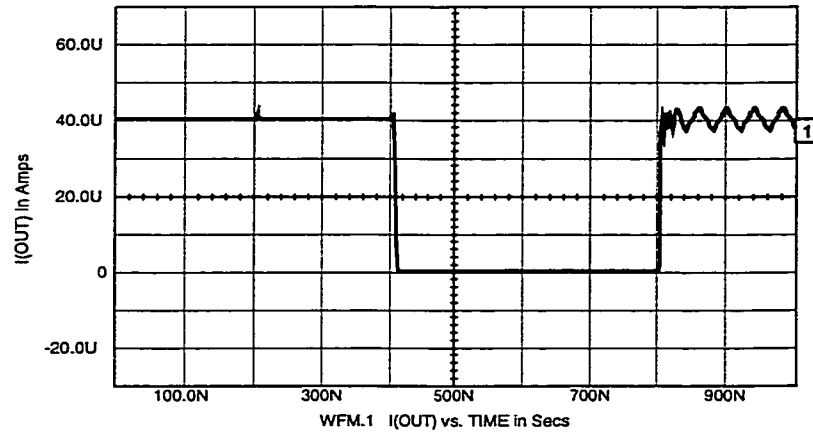
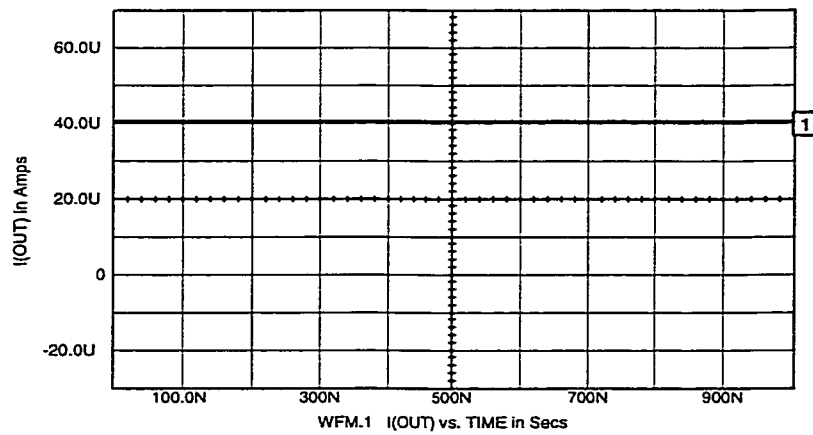


Figure 5.9: Complement of Literal transistor diagram.

Table 5.2 summarizes the simulation results obtained and the fault categories identified for the complement of literal circuit. Similar to the literal circuit, four

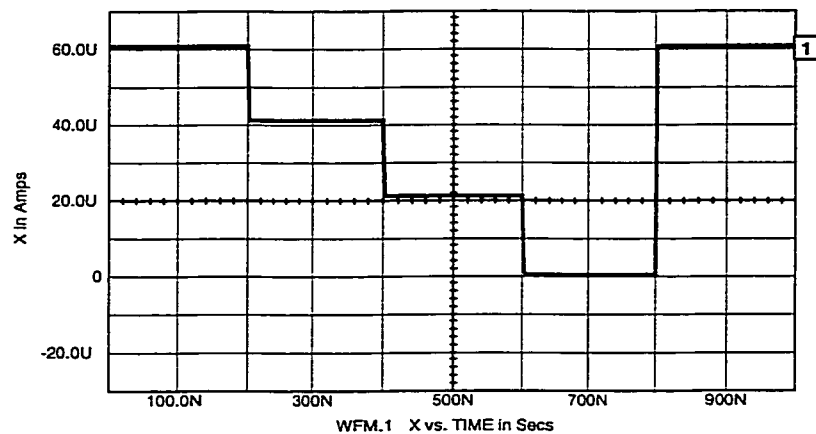


a. Fault free output

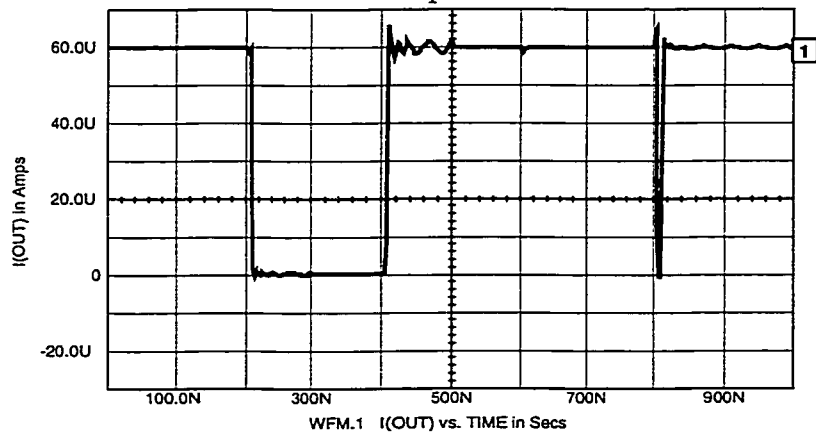


b. Output in the presence of S1gs

Figure 5.10: Response of the complement of literal, $\overline{2^2[x]^3}$.

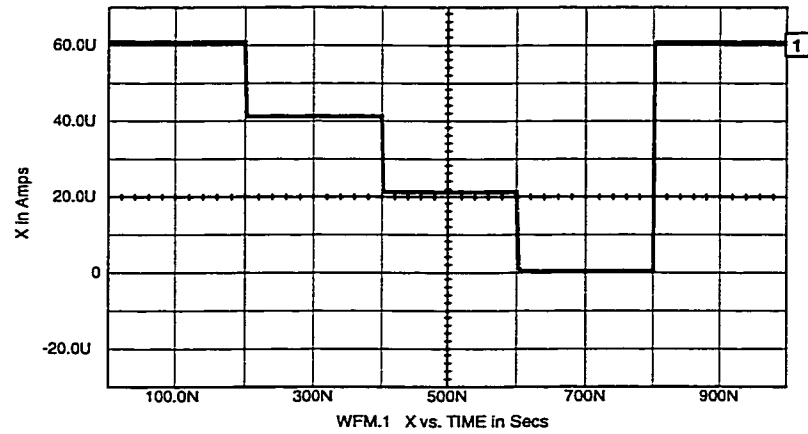


a. Input x

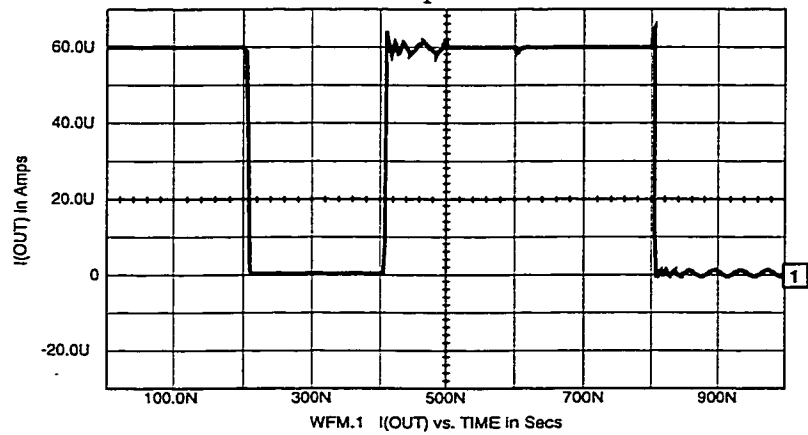


b. Fault free output

Figure 5.11: Response of the complement of literal, $\overline{3^2[x]^2}$.



a. Input x



b. Faulty output (sequential)

Figure 5.12: Response of the complement of literal, $\overline{3^2[x]^2}$ in the presence O10d.

main fault categories were found: sequential, SA0, SAk, and functional. Faults resulting in 3* output occurred at low percentage, 2.6% (faults S6sd and O6g output). The highest percentage of faults, 35.1%, resulted in stuck-at k output (SAk), opposite to the literal since most faults which caused SA0 in the literal resulted in SAk in the complement of literal. Sequential behavior was observed in 33.8% of the cases. The functional faults are listed and classified according to the resultant faulty functions into: bSA3, aSA0, complement (resulted exactly into the complement of the fault-free circuit) and other functional (irregular faulty functions). These totaled in 15 functional faults, 19.5%. The *others* fault category (in Table 5.2) includes Short S6gd, which resulted in oscillations (between 0 to 60 uA) at the output, and short S5gd, which resulted in irregular functional change that can not be classified in any of the above categories.

5.4 Fault Characterization of the Cycle

Figure 5.13 shows the cycle representation (transistor level). The operation of the circuit can be described based on another definition form of the cycle operation [20]:

$$X \leftarrow m = \begin{cases} x+(r-m) & \text{if } x < m \\ x-m & \text{if } x \geq m \end{cases}$$

Fault Category	Faults	Count	Percentage
Sequential	O1g, O2g, O3g, O4g, O4s, O4d, O5g, O7g, O8g, O8s, O8d, O9g, O9s, O9d, O10g, O10s, O10d, O11g, O11s, O11d, O12g, O12s, O12d, O13g, O13s, O13d	26	33.8%
SAk	S1gs, S1sd, S2gs, S3gs, S3gd, S3sd, S4gs, S4gd, S4sd, S5gs, S6gs, S7sd, S8gs, S8gd, S9gs, S9sd, S10sd, S11sd, S12gs, S12gd, S13gs, O1s, O1d, O2s, O2d, O5s, O5d,	27	35.1%
SA0	S7gs, O6s, O6d, O7s, O7d	5	6.5%
Functional	<i>bSA3</i> : S5sd, S8sd, S9gd, S10gs, O3s, O3d	6	19.5%
	<i>aSA0</i> : S2sd, S2gd, S11gs	3	
	<i>complement</i> : S7gd	1	
	<i>other functional</i> : S10gd, S11gd, S12sd, S13gd, S13sd	5	
		total 15	
3* faults	S6sd(kSA3*), O6g(sequential*)	2	2.6%
Others	S5gd, S6gd	2	2.6%
		total 77	

Table 5.2: Fault characterization of the complement of literal circuit.

In Figure 5.13, transistor 5 sources the $(r - m)$ current value and transistor 8 sinks m current value. Only one path will be open at any given time. This is controlled by the two switches: transistor 6 (p-type) and transistor 7 (n-type) which are controlled by the logic value at node A (transistor 4 will detect if $x \geq m$ and charge A to binary 1). The two cascaded inverters (transistors: 11, 12, 13 and 14) are only to restore the exact binary value of A at B for proper operation of the two switches. For example, if $x < m$ then node A, and hence node B, value will be binary 0. This will open the path through transistors 5 and 6 allowing the value of $(r-m)$ to be added to the mirror of x (transistor 3) and forming to the output. If, on the other hand, $x \geq m$, the path through transistors 7 and 8 will open and subtracts the value of m from the mirror of x .

Similar to the previous circuits, shorts and opens were inserted and simulated. Examples are presented below.

Example 1: The short S7sd causes the path through transistors 7 and 8 to be permanently open. If $x \geq m$ when path is normally open in the fault free circuit, the output will show the correct value. When $x < m$, the short will subtract the value of m from the correct output (through path 5 and 6). So, the faulty output will be less by the value of m . Figure 5.14 compares the faulty output, in the presence of this short, with the correct output for the circuit X^{-1} . The same input sequence ($x = 0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 0$) was used. The faulty circuit showed the correct output

for values of $x \geq 1$. When x was 0, the faulty output was 2 (40 μA) which is less than the correct output 3 (60 μA) by 1 (the value of m).

Example 2: The open at transistor 13's drain is classified in the sequential faults category. In the presence of this fault, node B voltage is floating when its charging path (through transistor 13) is open (if node A voltage is binary 1). Figure 5.15 shows the input/output responses of the cycle X^{-2} . The faulty sequential behavior is shown in Figure 5.16 (the input is repeated for clarity). Two different output values (intervals: 0-200 ns and 800 - 1000 ns) are found for the same input value $x=3$.

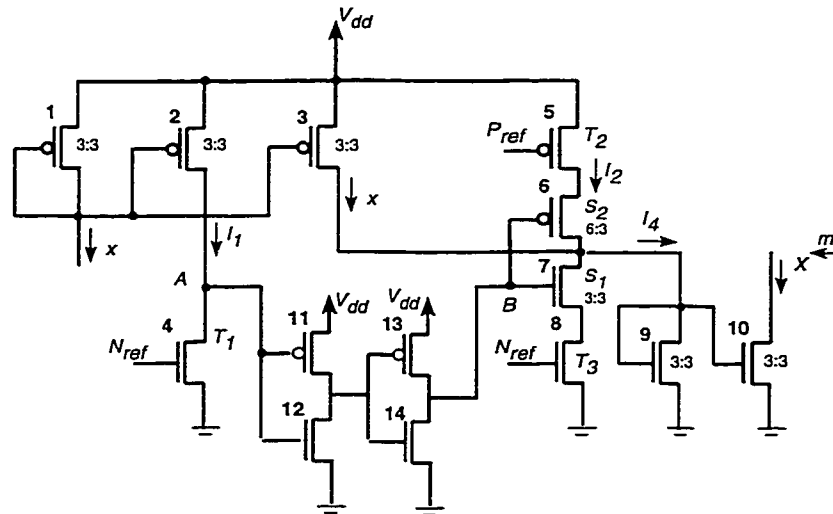
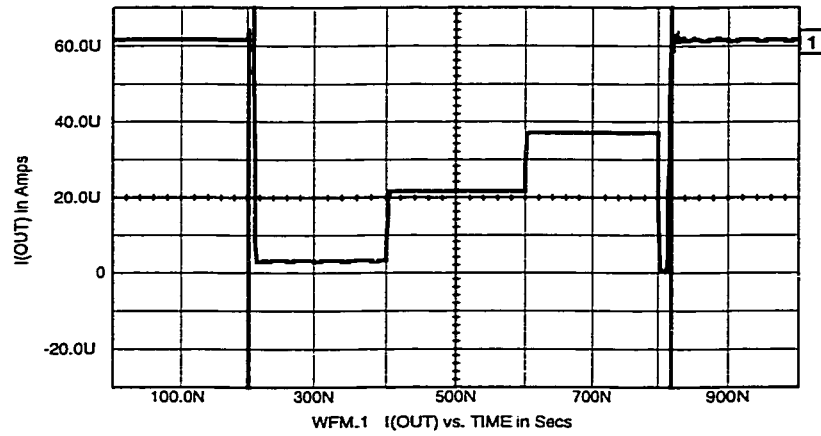
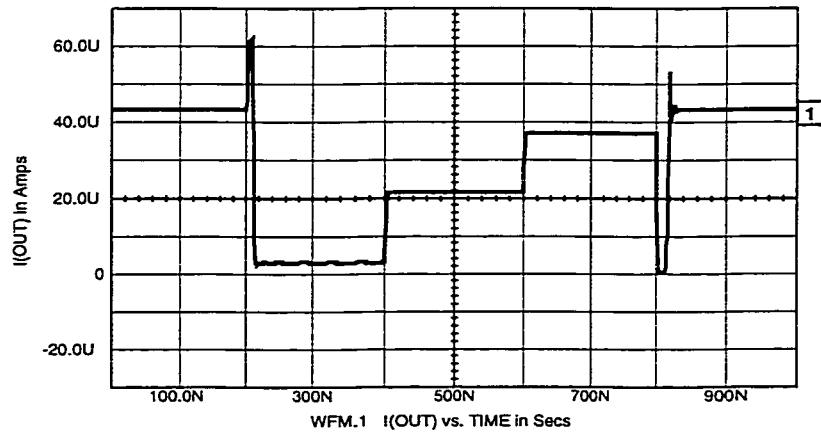


Figure 5.13: Cycle transistor diagram.

Characterization results are tabulated in Table 5.3. Four fault categories were found: functional, sequential SA3 and SA0. The highest fault percentage, 53.7%, are functional faults. This high percentage, as compared with the literal or its

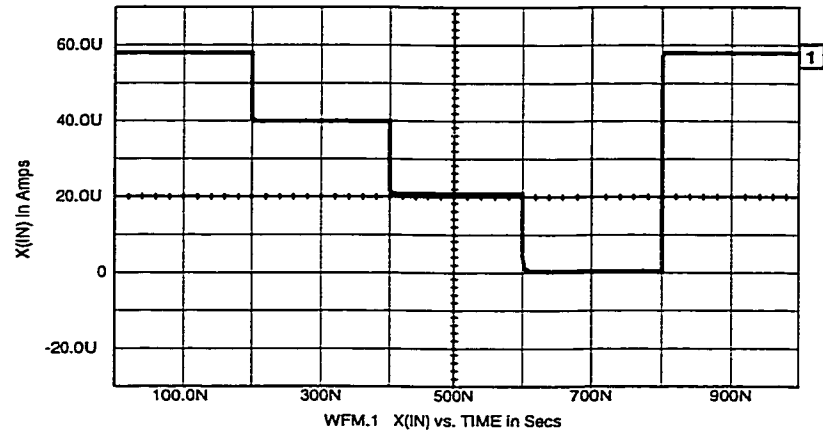


a. Fault free output

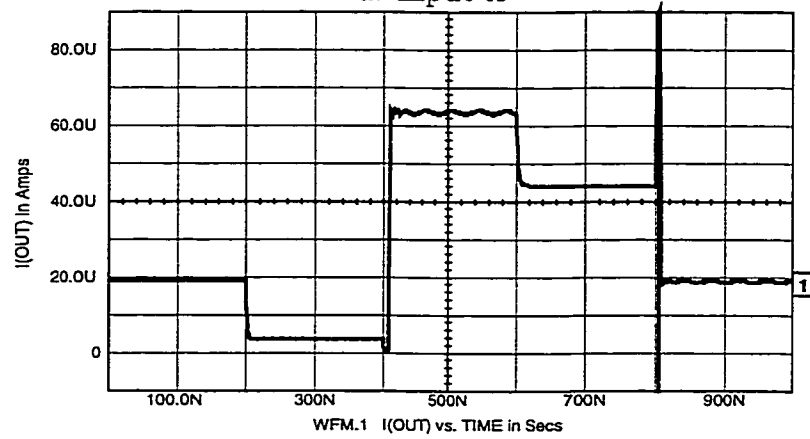


b. Output in the presence of S7sd

Figure 5.14: Response of the cycle, X^{-1} .

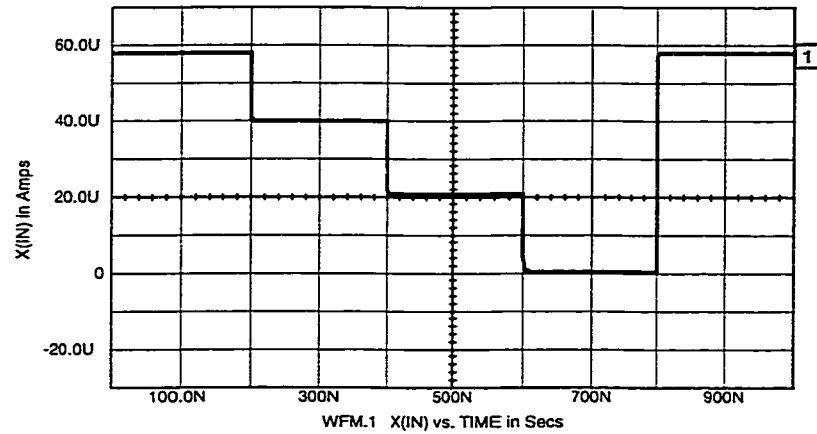


a. Input x

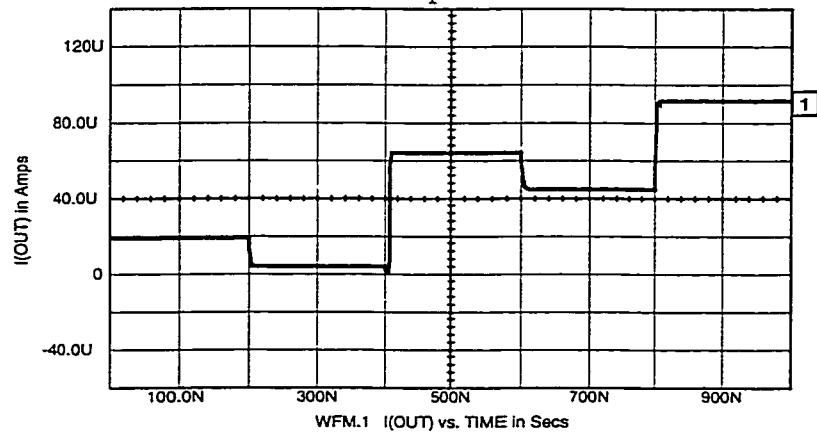


b. Fault free output

Figure 5.15: Response of the cycle, X^{-2} .



a. Input x



b. Faulty output (sequential)

Figure 5.16: Response of the cycle, X^{-2} in the presence of O13d.

complement, can be explained by the increased number of logic levels expected to appear at the cycle's output. The 4 levels (0, 1, 2 and 3) can appear at the cycle output while the literal shows only 2 logic levels, either 0 or k (the literal value). Sequential behavior was observed in 31.7% of the cases. Seven, 8.5%, faults are classified as SA3. This category is divided into SA3 and SA3* faults. The faulty circuit output in the presence of a SA3* fault is stuck-at 3* (values higher than 3). The rest of faults, 6.1%, are SA0 faults.

5.5 Fault Characterization of the *tSum*

Figure 5.17 shows the *tSum* representation. Similar to the cycle, there are two paths that determine the output and only one of which will be active at any given time. This is controlled by the inverter output (transistors 8 and 9) at point B which is connected to two switches S1 (n-type transistor 5) and S2 (p-type transistor 7). Point A, the inverter input, will charge to binary 1 if the sum of input currents $Y \geq 3$ (3 is r-1). This will drive node B to binary 0 closing S1 and opening S2. In this situation, the output is fed by transistor 6 which sources constant 60uA (logic 3). This will truncate the sum of inputs, Y, to 3 once its value exceeds 3. When $Y < 3$, the path through S1 will be enabled passing the sum value of the input currents.

Fault Category	Faults	Count	Percentage
Functional	S2sd, S2gd, S3gd, S4gs, S4sd, S4gd, S5gs, S5gd, S5sd, S6gs, S6gd, S6sd, S7gs, S7gd, S7sd, S8gs, S8gd, S8sd, S11gs, S11gd, S11sd, S12gs, S12gd, S12sd, S13gs, S13gd, S13sd, S14gs, S14gd, S14sd, O9s, O9d, O2s, O2d, O3s, O3d, O5s, O5d, O6s, O6d, O8s, O8d, O7s, O7d	44	53.7%
Sequential	O1g, O1s, O1d, O2g, O3g, O4g, O4s, O4d, O5g, O6g, O7g, O8g, O9g, O10g, O11g, O11s, O11d, O12g, O12s, O12d, O13g, O13s, O13d, O14g, O14s, O14d	26	31.7%
SA3	<i>SA3</i> : S1gs, S1sd, S2gs, S3gs	4	8.5%
	<i>SA3*</i> : S3sd, S10gd, S10sd	3	
		total 7	
SA0	S9gs, S9sd, S10gs, O10s, O10d	5	6.1%
		Total 82	

Table 5.3: Fault characterization of the cycle circuit.

Example 1: Short S4gd represents a class of faults that change the *tSum* into un-truncated sum function. The output will be the true sum of the inputs. In the presence of this short, the value of N_{ref} at the gate of transistor 4 will be higher than normal since it is shorted to the sum of the input currents. This will allow transistor 4 to pass higher currents which keeps Node A at binary 0. Node B will be at binary 1 all the time which results in connecting the output to the true, un-truncated, sum of the input currents. Figure 5.18 shows the normal and faulty outputs for this short. The two input currents used are $x_1 = 0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 0$ and $x_2 = 20\mu\text{A}$ (constant 1).

Example 2: The open at transistor 3's drain results in a functional fault. It blocks the sum value of the input currents (mirrored by transistor 3) from reaching the output. When the sum of input currents is less than 3, the faulty output will be 0. When the input currents are more than 3, the path through transistors 6 and 7 will be activated resulting in the truncated value at 3. Figure 5.19 shows the faulty output in the presence of this fault for the same inputs used in example 1 above.

Table 5.4 lists the fault categories for the *tSum* circuit. Four main fault categories are found: functional, SA3, sequential (and sequential*) and SA0. Similar to the cycle, the highest percentage occurred as functional faults. The faults listed under the sequential* category are sequential faults in nature but their faulty output will show 3* instead of 3. Sum* faults result in un-truncated sum function. The circuit

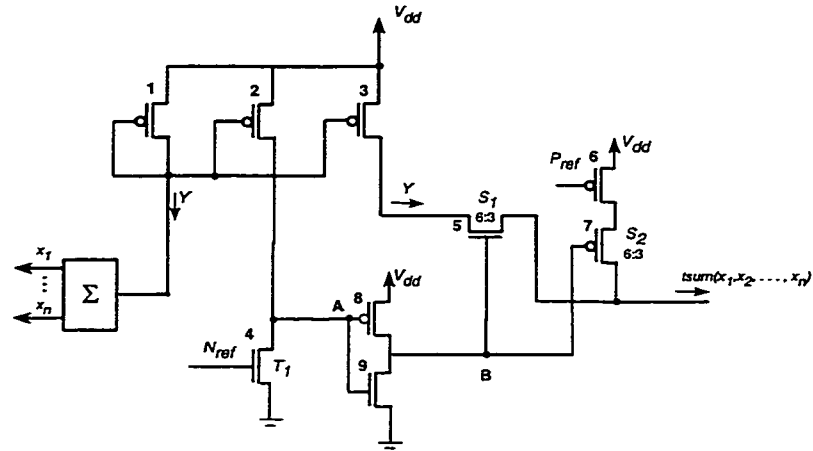
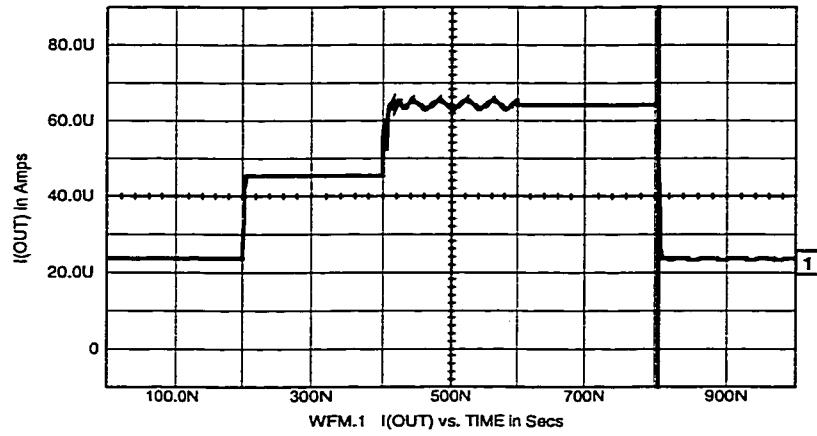


Figure 5.17: *tSum* transistor diagram.

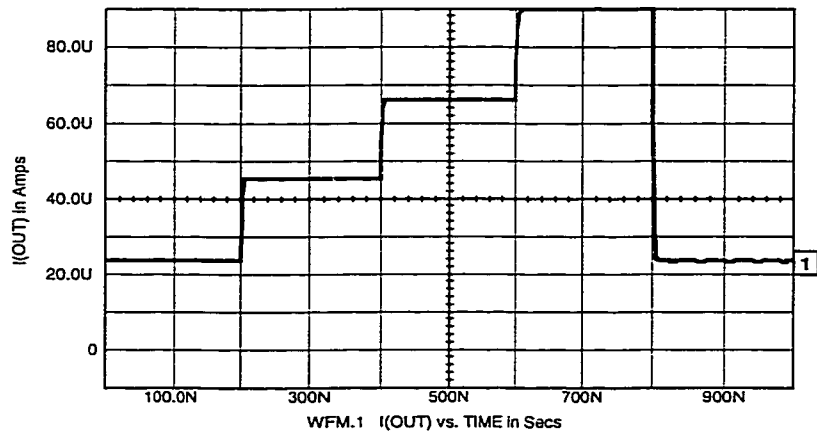
under these faults produces current levels equal to the actual sum. For example, if the inputs were 3 (60 uA) and 2 (40 uA), the output will show 100 uA. The test generation for such faults will be discussed in the next chapter. Also note that one single short (S3gd) results in high frequency oscillations spanning from 0 to 60uA at the output. S6gd short only affects the logic level 3. It becomes 2.5, i.e. 50uA instead of 3.

5.6 Fault Characterization of the *Min*

Figure 5.20 shows the *min* operator. It was designed using only current mode transistors. To describe the operation of the circuit, take two examples. If $x \leq y$, then $i_2 = i_1 - y = x - y = 0$. So, $i_3 = 0$ and $i_4 = x - i_3 = x - 0 = x$. The output, which is the mirror of i_4 will be x . On the other hand, if $x > y$, then



a. Fault free output



b. Output in the presence of S4gd

Figure 5.18: Response of a $tSum$ circuit.

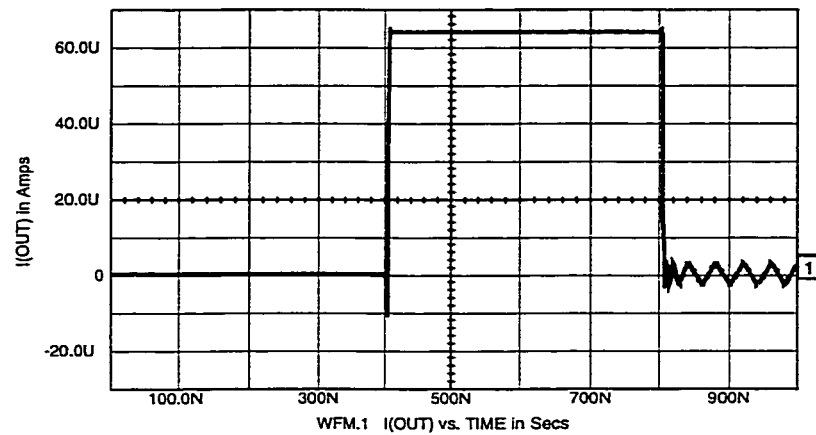


Figure 5.19: Response of a *tSum* circuit in the presence of O3d.

Fault Category	Faults	Count	Percentage
Functional	S5gs, S6gs, S7gs, S7gd, O3s, O3d, O5s, O5d, O6s, O6d, O7s, O7d <i>Sum</i> *: S4gd, S4sd, S5sd, S6sd, S8sd, S9gs, O2s, O2d	20	37.7%
SA3	S2gd, S2sd, S3sd, S4gs, S8gs, S9sd, O1s, O1d, O4s, O4d <i>SA3</i> *: S5gd, S7sd, S8gd, S9gd	14	26.4%
sequential	O1g, O2g, O3g, O4g, O5g, O7g, O8g, O8s, O8d, O9g	10	18.9%
Sequential*	O6g, O9s, O9d	3	5.7%
SA0	S1gs, S1sd, S2gs, S3gs	4	7.5%
Others	S3gd (High freq. Osc.) S6gd (3 becomes 2.5)	2	3.8%
		total 53	

Table 5.4: Fault characterization of the *tSum* circuit.

$i_2 = (x - y) = i_3$. So, $i_4 = x - i_3 = x - (x - y) = y$ and the output will be y .

Example 1: The short S7gs resulted in a SA0 fault. It raises the gate voltage of transistor 7 to Vdd value which closes the path for the output. Short 7gd on resulted in a SA3 fault. Transistor 7 gate voltage under this short is lower than normal. This allows more current (even more than 6uA) to pass through all the time and independent of the inputs. Figure 5.21 shows the normal and faulty outputs in the presence of the short 7gd. The inputs used were: $y = \text{constant } 2$ and $x = 0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 0$.

Example 2: Figure 5.22 shows the faulty outputs in the presence of the Open at transistor 5's source (O5s). The subtraction process by the current i_3 (Figure 5.20) is disabled and hence the faulty output follows the input x regardless of the other input (y) value.

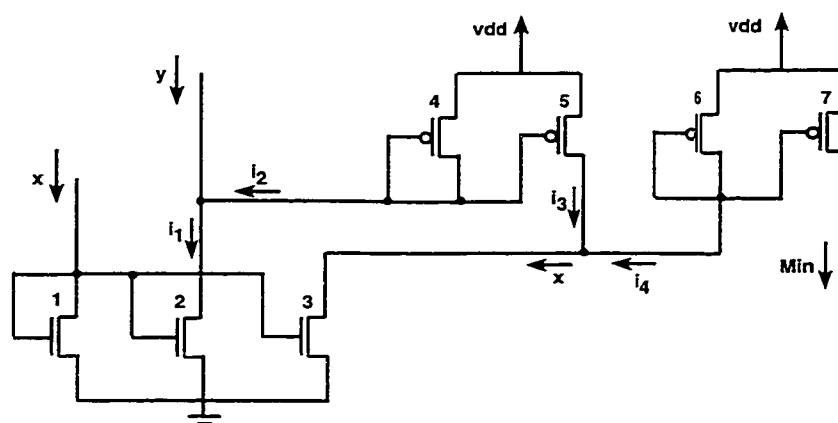
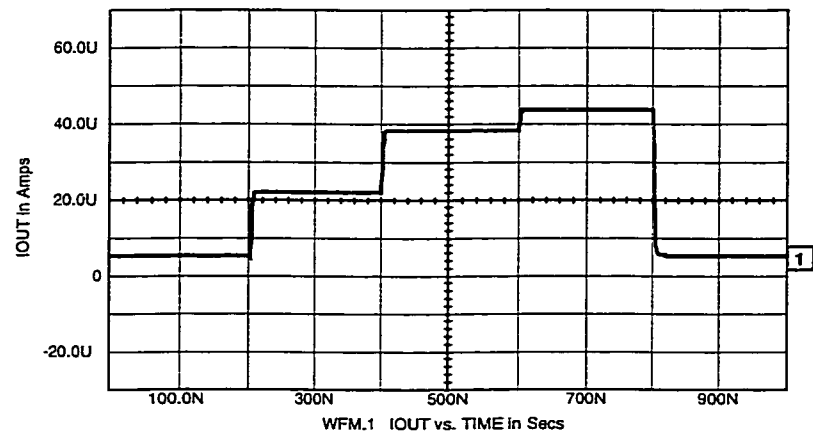
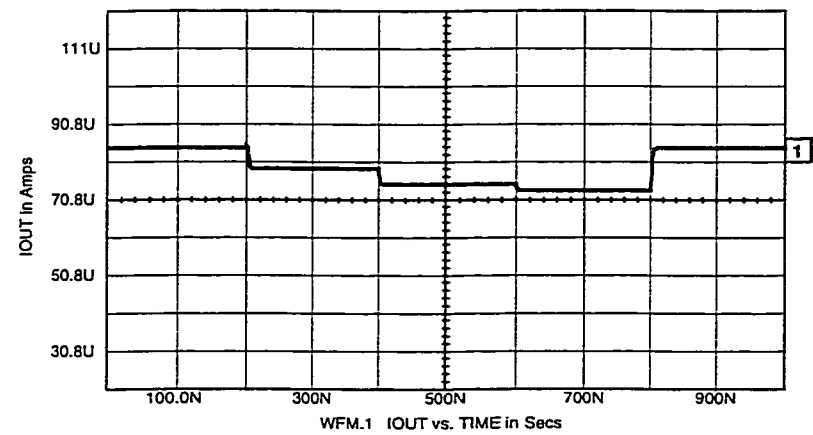


Figure 5.20: *Min* transistor diagram.



a. Fault free output



b. Output in the presence of S7gd

Figure 5.21: Response of a *min* circuit.

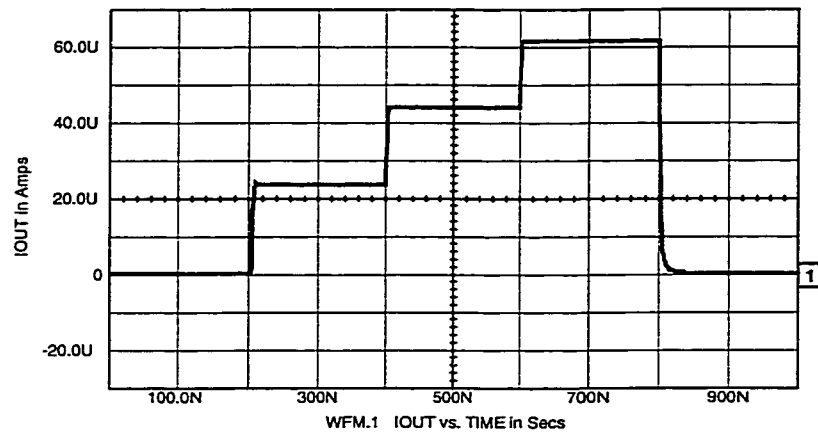


Figure 5.22: Response of a *min* circuit in the presence of O5s.

Table 5.5 lists the simulation results. Four fault categories are characterized for the *min*: SA0, SA3, functional and sequential. The highest percentage (33.3%) was SA0 faults. Functional faults are encountered in 30.8% of the cases. Sequential faults occurrence is also at high percentage 28.2%. The lowest occurrence, 7.7%, is for the SA3 faults.

Fault Category	Faults	Count	Percentage
SA0	S1gs, S1sd, S2gs, S2sd, S3gs, S5sd, S6gs, S6sd, S7gs, O3s, O3d, O7s, O7d	13	33.3%
SA3	S3sd, S7gd, S7sd	3	7.7%
Functional	S2gd, S3gd, S4gs, S4sd, S5gs, S5gd, O2s, O2d, O4s, O4d, O5s, O5d	12	30.8%
Sequential	O1g, O1s, O1d, O2g, O3g, O4g, O5g, O6g, O6s, O6d, O7g	11	28.2%
		total 39	

Table 5.5: Fault characterization of the *min* circuit.

5.7 Fault Characterization Summary

The fault characterization conducted on the MVL set in this work resulted in 4 main fault categories:

1. **Sequential faults:** which have sequential behavior.
2. **SA0:** the output is stuck-at zero level.
3. **SA3:** the output is stuck-at level 3. In the case of literals, this appeared as stuck at the literal value (the value of k).
4. **Functional faults:** which change the circuit function into a different one from what it was designed for.

In addition, some other peculiar faults were found with low percentages. The percentages of each category against the circuits studied are summarized in Table 5.6.

Fault category	Literal	Comp. of Literal	Cycle	tSum	Min
Sequential	33.8%	33.8%	31.7%	24.6%	28.2%
Functional	20.8%	20.8%	53.7%	37.7%	30.8%
SA0	40.3%	6.5%	6.1%	7.5%	33.3%
SA3	1.3%	35.1%	8.5%	26.4%	7.7%
Others	3.9%	3.9%		3.8%	

Table 5.6: Summary of the characterized fault categories.

Chapter 6

Testing of the MVL Set

For a given circuit under test, the aim of any test generation process is to find the proper inputs that have their fault-free outputs different from the faulty output(s). Once these inputs are applied to the circuit, they will produce noticeable wrong output(s), which are different from the fault free output(s), if the circuit is faulty. For example, the fault-free output for x^{-1} , when $x = 0$, is 3. To test for SA0 fault at the output, 0 is a valid test because if it is applied, it will produce a fault-free output of 3 which is different from the faulty output (0). On the other hand, $x = 1$ is not a valid test because if it is applied, it will produce a fault-free output of 0 which is not distinguishable from the faulty output. Sequential faults need two test vectors since the circuit output under such faults depends on both the previous

state and the current input. One vector is needed to initialize the circuit and the second vector to excite the fault.

For each of the circuits studied, test vectors were generated by simulation of each fault against all possible input variations (a sequence of two different inputs for sequential faults and single input for other faults). In the following sections, lists of the valid tests for every circuit under all possible faults (shorts and opens) are provided. From these lists, minimal sets or input sequences that test for all faults are extracted. A special testing procedure for the peculiar 3* faults will be presented at the end of the chapter.

6.1 Testing of the Literal

Table 6.1 lists the faults and their tests for the literal operator. The table is divided into six main columns each for one instance of the literal, namely $k^1[x]^1$, $k^1[x]^2$, $k^1[x]^3$, $k^2[x]^2$, $k^2[x]^3$, $k^3[x]^3$. Other possible configurations, like $k^0[x]^1$, can be represented by an equivalent complement of literal instance [20, 21].

For each single fault (or two equivalent faults), there is one corresponding entry. Valid test vectors are listed along the rows. These tests are obtained and verified by simulation. Sequential faults have their tests listed as pairs of the initializing input

and the test input as (x_1, x_2) , where x_1 is the first input which should be followed by x_2 to excite the fault. Functional and other non-sequential faults need only one input test. Valid tests are listed separated by commas. Entries marked with n/a are for faults that resulted in fault-free behavior. For example, the fault O4d results in a sequential behavior in the $k^2[x]^2$ circuit. The test $(3, 1)$ is listed as a valid test. First, the circuit is initialized by the input $x_1 = 3$ and then $x_2 = 1$ is applied to excite the fault and detect a faulty value at the output. Other (non-sequential) faults can be tested by a single input value. To test for a SAK fault in the $k^1[x]^1$ circuit, the input value $x = 0$ can be applied. The fault free output in this case is 0 and the faulty output is k.

In order to find the minimal test set(s) / sequences, we have applied the concept of fault coverage table. In a fault coverage table, each test vector is represented by a row and each fault by a column. If a test detects a given fault, the corresponding row-column intersection is checked (marked). For example, Table 6.2(a) shows the fault coverage table for the non-sequential faults for $k^1[x]^1$. The fault coverage table can be simplified according to the following rules:

1. If a column A is included in column B, then column B can be deleted. Column B includes column A if all A's checks (marks) are included in B. This indicates that picking any test for A will definitely test for B. So, no need to list column B.

2. If row A is included in row B, then row A can be deleted. All faults covered by A are also covered by B. So, B can be used instead.
3. If a column has only one location checked, it indicates that the corresponding test vector is essential (must be included in any test sequence).

Table 6.2(a) is obtained directly from the non-sequential part (single tests) in Table 6.1. After applying column simplification, the simplified coverage table for $k^1[x]^1$ is listed in Table 6.2(b). Column bSA3, in Table 6.2(a), is included in columns SAk, S5gd and S13gd/S10sd and so they are deleted and only column bSA3 appears in Table 6.2(b). Similarly, column SA0 is included in column S6sd and hence only column SA0 appears in Table 6.2(b). Column aSA0 is essential and is also included in Table 6.2(b). From the simplified fault coverage Table 6.2(b), it is found that any test sequence must include 0 and 1. Also, either 2 or 3 must be included. This can be written as: 0, 1, [2 (3)]. Required tests are listed separated by commas. When there is a choice between a number of tests, they will be placed inside square brackets [] separated by the small brackets '()'. This notation will be used throughout the next sections. Sequential tests have two values for a single vector. These will be listed as pairs of (x1, x2). Table 6.2(c) lists the simplified coverage table for the sequential faults occurring in $k^1[x]^1$. The tests (0, 1), (1, 0) must be included. Either (1, 2) or (1, 3) must be included too. Finally, one of the test (2, 1) or (3, 1) must be included. This can be written as:

$(0, 1), (1, 0), [(1, 2(3)), [2(3), 1]$.

Combining tests obtained from sequential and non-sequential faults $(0, 1, [2(3)]$ and $(0, 1), (1, 0), [(1, 2(3)), [2(3), 1])$, the following minimal sets of vectors are obtained:

$\{(0, 1), (1, 0), (1, 2), (2, 1)\}$ or

$\{(0, 1), (1, 0), (1, 3), (3, 1)\}$

To obtain the minimal test sequence, the above pairs have to be ordered to obtain the shortest possible sequences. For the above sets, the minimal test sequences that test for all faults are:

$x = 1 \rightarrow 0 \rightarrow 1 \rightarrow 2 \rightarrow 1$ or

$x = 1 \rightarrow 0 \rightarrow 1 \rightarrow 3 \rightarrow 1$

Note: other possible sequences can be obtained but they are similar and of the same size. They are

$x = 1 \rightarrow 2 \rightarrow 1 \rightarrow 0 \rightarrow 1$ or

$x = 1 \rightarrow 3 \rightarrow 1 \rightarrow 0 \rightarrow 1$

Following similar steps and simplifying coverage tables, the tests for the rest of the instances can be written as in Table 6.3. The resultant minimal test sets and minimal test sequences are listed in Table 6.4.

Fault	$k^1[x]^1$	$k^1[x]^2$	$k^1[x]^3$	$k^2[x]^2$	$k^2[x]^3$	$k^3[x]^3$
O1g	(0, 1) (2, 1) (3, 1)	(0, 1) (0, 2) (2, 1) (3, 1)	(0, 1) (0, 2) (0, 3) (1, 2) (1, 3) (2, 1) (2, 3) (3, 1) (3, 2)	(0, 2) (1, 2) (3, 2)	(0, 2) (0, 3) (1, 2) (1, 3)	(0, 3) (1, 3) (2, 3)
O2g	(0, 1) (1, 0) (2, 0) (3, 0)	(0, 1) (0, 2) (1, 0) (2, 0) (3, 0)	(0, 1) (0, 2) (0, 3) (1, 0) (2, 0) (3, 0)	(0, 2) (1, 2) (2, 0) (2, 1) (3, 0) (3, 1)	(0, 2) (0, 3) (1, 2) (1, 3) (2, 0) (2, 1) (3, 0) (3, 1)	(0, 3) (1, 3) (2, 3) (3, 0) (3, 1) (3, 2)
O3g	(0, 2) (0, 3) (1, 2) (1, 3) (2, 1) (3, 1)	(0, 3) (1, 3) (2, 3) (3, 1) (3, 2)	n/a	(0, 3) (1, 3) (2, 3) (3, 2)	n/a	n/a
O4s/ O4d	(1, 0) (2, 0) (3, 0)	(1, 0) (2, 0) (3, 0)	(1, 0) (2, 0) (3, 0)	(1, 0) (2, 0) (3, 0) (2, 1) (3, 1)	(1, 0) (2, 0) (3, 0) (2, 1) (3, 1)	(1, 0) (2, 0) (3, 0) (2, 1) (3, 1) (1, 2) (3, 2)
O4g	n/a	n/a	n/a	(0, 2) (1, 2) (2, 1) (3, 1)	(0, 2) (0, 3) (1, 2) (2, 1) (3, 1)	(0, 3) (1, 3) (2, 3) (3, 2)
O5g	(0, 2) (0, 3) (1, 2) (2, 1) (3, 1)	(0, 3) (1, 3) (2, 3) (3, 2)	n/a	(0, 3) (1, 3) (2, 3) (3, 2)	n/a	n/a
O7g	(1, 0) (1, 2) (1, 3) (0, 1) (2, 1) (3, 1)	(1, 0) (1, 3) (2, 0) (2, 3) (0, 1) (0, 2) (3, 1) (3, 2)	(1, 0) (2, 0) (3, 0) (0, 1) (0, 2) (0, 3)	(2, 0) (2, 1) (2, 3) (0, 2) (1, 2) (3, 2)	(2, 0) (2, 1) (3, 0) (3, 1) (0, 2) (0, 3) (1, 2) (1, 3)	(3, 0) (3, 1) (3, 2) (0, 3) (1, 3) (2, 3)
O8s/ O8d	(1, 0) (2, 0) (3, 0)	(1, 0) (2, 0) (3, 0)	(1, 0) (2, 0) (3, 0)	(2, 0) (2, 1) (3, 0) (3, 1)	(2, 0) (2, 1) (3, 0) (3, 1)	(3, 0) (3, 1) (3, 2)
O8g	(0, 1) (1, 0) (2, 0) (3, 0)	(0, 1) (0, 2) (1, 0) (2, 0) (3, 0)	(0, 1) (0, 2) (0, 3) (1, 0) (2, 0) (3, 0)	(0, 2) (1, 2) (2, 0) (2, 1) (3, 0) (3, 1)	(0, 2) (0, 3) (1, 2) (1, 3) (2, 0) (2, 1) (3, 0) (3, 1)	(0, 3) (1, 3) (2, 3) (3, 0) (3, 1) (3, 2)
O9s/ O9d	(0, 1)	(0, 1) (0, 2)	(0, 1) (0, 2) (0, 3)	(0, 2) (1, 2)	(0, 2) (0, 3) (1, 2) (1, 3)	(0, 3) (1, 3) (2, 3)
O9g	(0, 1)	(0, 1) (0, 2)	(0, 1) (0, 2) (0, 3)	(0, 2) (1, 2)	(0, 2) (0, 3) (1, 2) (1, 3)	(0, 3) (1, 3) (2, 3)

Table 6.1: Exhaustive test list for the literal circuit

Fault	$k^1[x]^1$	$k^1[x]^2$	$k^1[x]^3$	$k^2[x]^2$	$k^2[x]^3$	$k^3[x]^3$
O10s/ O10d	(0, 1) (2, 1) (3, 1)	(0, 1) (0, 2) (3, 1) (3, 2)	(0, 1) (0, 2) (0, 3)	(0, 2) (1, 2) (3, 2)	(0, 2) (0, 3) (1, 2) (1, 3)	(0, 3) (1, 3) (2, 3)
O10g	(2, 1) (3, 1)	(3, 1) (3, 2)	n/a	(3, 2)	n/a	n/a
O11s/ O11d	(0, 1) (2, 1) (3, 1)	(0, 1) (0, 2) (3, 1) (3, 2)	(0, 1) (0, 2) (0, 3)	(0, 2) (1, 2) (3, 2)	(0, 2) (0, 3) (1, 2) (1, 3)	(0, 3) (1, 3) (2, 3)
O11g	(0, 1)	(0, 1) (0, 2)	(0, 1) (0, 2) (0, 3)	(0, 2) (1, 2)	(0, 2) (0, 3) (1, 2) (1, 3)	(0, 3) (1, 3) (2, 3)
O12S/ O12d	(1, 0)	(1, 0) (2, 0)	(1, 0) (2, 0) (3, 0)	(2, 0) (2, 1)	(2, 0) (2, 1) (3, 0) (3, 1)	(3, 0) (3, 1) (3, 2)
O12g	(0, 1) (1, 0)	(0, 1) (0, 2) (1, 0) (2, 0)	(0, 1) (0, 2) (0, 3) (1, 0) (2, 0) (3, 0)	(0, 2) (1, 2) (2, 0) (2, 1)	(0, 2) (0, 3) (1, 2) (1, 3) (2, 0) (2, 1) (3, 0) (3, 1)	(0, 3) (1, 3) (2, 3) (3, 0) (3, 1) (3, 2)
O13s/ O13d	(1, 2) (1, 3)	(1, 3) (2, 3)	n/a	(2, 3)	n/a	n/a
O13g	(1, 2) (1, 3) (2, 1) (3, 1)	(1, 3) (2, 3) (3, 1) (3, 2)	n/a	(2, 3) (3, 2)	n/a	n/a
O6g (k*)	(0, 1) (2, 1) (3, 1)	(0, 1) (0, 2) (3, 1) (3, 2)	(0, 1) (0, 2) (0, 3)	(0, 2) (1, 2) (3, 2)	(0, 2) (0, 3) (1, 2) (1, 3)	(0, 3) (1, 3) (2, 3)
SA0	1	1, 2	1, 2, 3	2	2, 3	3
SAk	0, 2, 3	0, 3	0	0, 1, 3	0, 1	0, 1, 2
aSA0	0	0	0	0, 1	0, 1	0, 1, 2
bSA3	2, 3	3	n/a	3	n/a	n/a
S5gd	2, 3	3	Para_k at: all 1, 2 & 3	1, 3	1	2
S13gd/ S10sd	2, 3	3	Para_k at: all 1, 2 & 3	3	Para_k at: all 2 & 3	Para_k at: 3
S6sd	k* at: 1	k* at: 1 or 2	k* at: 1, 2 or 3	k* at: 2	k* at: 2 or 3	k* at: 3

Table 6.1 Exhaustive test list for the literal circuit (continued)

Test vector	SA0	SAk	aSA0	bSA3	S5gd	S13gd/S10sd	S6sd
0		X	X				
1	X						X
2		X		X	X	X	
3		X		X	X	X	

a. non-sequential faults

	SA0	bSA3	aSA0
0			X
1	X		
2		X	
3		X	

b. non-sequential faults (simplified)

	O9s/d	O10g	O12s/d	O13s/d
0 → 1	X			
0 → 2				
0 → 3				
1 → 0			X	
1 → 2				X
1 → 3				X
2 → 0				
2 → 1		X		
2 → 3				
3 → 0				
3 → 1		X		
3 → 2				

c. sequential faults (simplified)

Table 6.2: Fault coverage tables for $k^1[x]^1$

Literal	Test
$k^1[x]^1$	sequential: (0, 1), (1, 0), [(1, 2(3)), [2(3), 1]
	non-sequential: 0, 1, [2 (3)]
$k^1[x]^2$	sequential: [0, 1(2)], [1(2), 0], [1(2), 3], [3, 1(2)]
	non-sequential: 0, 3, [1(3)]
$k^1[x]^3$	sequential: [(0, 1(2)(3)), [1(2)(3), 0]
	non-sequential: 0, 1, 2, 3
$k^2[x]^2$	sequential: (3, 2), (2, 3), [2, 0(1)], [0(1), 2]
	non-sequential: 2, 3, [0(1)]
$k^2[x]^3$	sequential: [(0, 2(3)) (1, 2(3))], [(2, 0(1)) (3, 0(1))], [(0, 2(3)) (1, 2) (2(3), 1)]
	non-sequential: 1, 2, 3
$k^3[x]^3$	sequential: [0(1)(2), 3], [3, 0(1)(2)]
	non-sequential: 2, 3, [0(1)]

Table 6.3: Literal tests.

6.2 Testing of the Complement of Literal

Table 6.5 shows the test vectors for the complement of literal faults. The fault coverage table method and similar steps to those used for the literal resulted in the complement of literal tests as listed in Table 6.6. The resultant minimal test sets and sequences are listed in Table 6.7.

6.3 Testing of the Cycle

Table 6.8 shows the test vectors for the cycle operator. The simplified test lists obtained by simplifying the coverage tables are listed in Table 6.9. The resultant

Literal	Test Sequence
$k^1[x]^1$	$\{(0, 1), (1, 0), (1, 2), (2, 1)\}$ or $\{(0, 1), (1, 0), (1, 3), (3, 1)\}$ Minimal test sequence: $1 \rightarrow 0 \rightarrow 1 \rightarrow 2 \rightarrow 1$ or $1 \rightarrow 0 \rightarrow 1 \rightarrow 3 \rightarrow 1$
$k^1[x]^2$	$\{(0, 1), (1, 0), (1, 3), (3, 1)\}$ or $\{(0, 2), (2, 0), (2, 3), (3, 2)\}$ Minimal test sequence: $1 \rightarrow 0 \rightarrow 1 \rightarrow 3 \rightarrow 1$ or $2 \rightarrow 0 \rightarrow 2 \rightarrow 3 \rightarrow 2$
$k^1[x]^3$	$\{(0, 3), (1, 0), 2\}$ or $\{(1, 0), (0, 2), 3\}$ or $\{(2, 0), (0, 1), 3\}$ $\{(2, 0), (0, 3), 1\}$ or $\{(3, 0), (0, 1), 2\}$ or $\{(3, 0), (0, 2), 1\}$ Minimal test sequence: $1 \rightarrow 0 \rightarrow 3 \rightarrow 2$ or $1 \rightarrow 0 \rightarrow 2 \rightarrow 3$ or $2 \rightarrow 0 \rightarrow 1 \rightarrow 3$ or $2 \rightarrow 0 \rightarrow 3 \rightarrow 1$ or $3 \rightarrow 0 \rightarrow 1 \rightarrow 2$ or $3 \rightarrow 0 \rightarrow 2 \rightarrow 1$
$k^2[x]^2$	$\{(3, 2), (2, 3), (2, 0), (0, 2)\}$ or $\{(3, 2), (2, 3), (2, 1), (1, 2)\}$ Minimal test sequence: $2 \rightarrow 3 \rightarrow 2 \rightarrow 0 \rightarrow 2$ or $2 \rightarrow 3 \rightarrow 2 \rightarrow 1 \rightarrow 2$
$k^2[x]^3$	$\{(1, 2), (3, 1)\}$ or $\{(1, 3), (2, 1)\}$ Minimal test sequence: $3 \rightarrow 1 \rightarrow 2$ or $2 \rightarrow 1 \rightarrow 3$
$k^3[x]^3$	$\{(2, 3), (3, 2)\}$ Minimal test sequence: $2 \rightarrow 3 \rightarrow 2$

Table 6.4: Minimal test sets/sequences for the literal.

Fault	$\overline{k^1[x]^1}$	$\overline{k^1[x]^2}$	$\overline{k^1[x]^3}$	$\overline{k^2[x]^2}$	$\overline{k^2[x]^3}$	$\overline{k^3[x]^3}$
O1g	(0, 1) (2, 1) (3, 1)	(0, 1) (0, 2) (2, 1) (3, 1)	(0, 1) (0, 2) (0, 3) (1, 2) (1, 3) (2, 1) (2, 3) (3, 1) (3, 2)	(0, 2) (1, 2) (3, 2)	(0, 2) (0, 3) (1, 2) (1, 3)	(0, 3) (1, 3) (2, 3)
O2g	(0, 1) (1, 0) (2, 0) (3, 0)	(0, 1) (0, 2) (1, 0) (2, 0) (3, 0)	(0, 1) (0, 2) (0, 3) (1, 0) (2, 0) (3, 0)	(0, 2) (1, 2) (2, 0) (2, 1) (3, 0) (3, 1)	(0, 2) (0, 3) (1, 2) (1, 3) (2, 0) (2, 1) (3, 0) (3, 1)	(0, 3) (1, 3) (2, 3) (3, 0) (3, 1) (3, 2)
O3g	(0, 2) (0, 3) (1, 2) (1, 3) (2, 1) (3, 1)	(0, 3) (1, 3) (2, 3) 3, 1) (3, 2)	n/a	(0, 3) (1, 3) (2, 3) (3, 2)	n/a	n/a
O4s/ O4d	(1, 0) (2, 0) (3, 0)	(1, 0) (2, 0) (3, 0)	(1, 0) (2, 0) (3, 0)	(1, 0) (2, 0) (3, 0) (2, 1) (3, 1)	(1, 0) (2, 0) (3, 0) (2, 1) (3, 1)	(1, 0) (2, 0) (3, 0) (2, 1) (3, 1) (1, 2) (3, 2)
O4g	n/a	n/a	n/a	(0, 2) (1, 2) (2, 1) (3, 1)	(0, 2) (0, 3) (1, 2) (2, 1) (3, 1)	(0, 3) (1, 3) (2, 3) (3, 2)
O5g	(0, 2) (0, 3) (1, 2) (2, 1) (3, 1)	(0, 3) (1, 3) (2, 3) (3, 2)	n/a	(0, 3) (1, 3) (2, 3) (3, 2)	n/a	n/a
O7g	(1, 0) (1, 2) (1, 3) (0, 1) (2, 1) (3, 1)	(1, 0) (1, 3) (2, 0) (2, 3) (0, 1) (0, 2) (3, 1) (3, 2)	(1, 0) (2, 0) (3, 0) (0, 1) (0, 2) (0, 3)	(2, 0) (2, 1) (2, 3) (0, 2) (1, 2) (3, 2)	(2, 0) (2, 1) (3, 0) (3, 1) (0, 2) (0, 3) (1, 2) (1, 3)	(3, 0) (3, 1) (3, 2) (0, 3) (1, 3) (2, 3)
O8s/ O8d	(2, 1) (3, 1)	(3, 1) (3, 2)	n/a	(3, 2)	n/a	n/a
O8g	(2, 1) (3, 1)	(3, 1) (3, 2)	n/a	(3, 2)	n/a	n/a
O9s/ O9d	(0, 2) (0, 3) (1, 2) (1, 3)	(0, 3) (1, 3) (2, 3)	n/a	(0, 3) (1, 3) (2, 3)	n/a	n/a
O9g	(0, 2) (0, 3) (1, 2) (1, 3)	(0, 3) (1, 3) (2, 3)	n/a	(0, 3) (1, 3) (2, 3)	n/a	n/a

Table 6.5: Exhaustive test list for the complement of literal.

Fault	$\overline{k^1[x]^1}$	$\overline{k^1[x]^2}$	$\overline{k^1[x]^3}$	$\overline{k^2[x]^2}$	$\overline{k^2[x]^3}$	$\overline{k^3[x]^3}$
O10s/ O10d	(1, 2) (1, 3)	(1, 3) (2, 3)	n/a	(2, 3)	n/a	n/a
O10g	(1, 2) (1, 3)	(1, 3) (2, 3)	n/a	(2, 3)	n/a	n/a
O11s/ O11d	(1, 0)	(1, 0) (2, 0)	(1, 0) (2, 0) (3, 0)	(2, 0) (2, 1)	(2, 0) (2, 1) (3, 0) (3, 1)	(3, 0) (3, 1) (3, 2)
O11g	(1, 0)	(1, 0) (2, 0)	(1, 0) (2, 0) (3, 0)	(2, 0) (2, 1)	(2, 0) (2, 1) (3, 0) (3, 1)	(3, 0) (3, 1) (3, 2)
O12s/ O12d	(0, 1) (2, 1) (3, 1)	(0, 1) (0, 2) (3, 1) (3, 2)	(0, 1) (0, 2) (0, 3)	(0, 2) (1, 2) (3, 2)	(0, 2) (0, 3) (1, 2) (1, 3)	(0, 3) (1, 3) (2, 3)
O12g	(0, 1)	(0, 1) (0, 2)	(0, 1) (0, 2) (0, 3)	(0, 2) (1, 2)	(0, 2) (0, 3) (1, 2) (1, 3)	(0, 3) (1, 3) (2, 3)
O13s/ O13d	(0, 1) (2, 1) (3, 1)	(0, 1) (0, 2) (3, 1) (3, 2)	(0, 1) (0, 2) (0, 3)	(0, 2) (1, 2) (3, 2)	(0, 2) (0, 3) (1, 2) (1, 3)	(0, 3) (1, 3) (2, 3)
O13g	(2, 1) (3, 1)	(3, 1) (3, 2)	n/a	(3, 2)	n/a	n/a
O6g (k*)	(1, 0) (1, 2) (1, 3)	(1, 0) (1, 3) (2, 0) (2, 3)	(1, 0) (2, 0) (3, 0)	(2, 0) (2, 1) (2, 3)	(2, 0) (2, 1) (3, 0) (3, 1)	(3, 0) (3, 1) (3, 2)
SA0	0, 2, 3	0, 3	0	0, 1, 3	0, 1	0, 1, 2
SAk	1	1, 2	1, 2, 3	2	2, 3	3
aSA0	0	0	0	0, 1	0, 1	0, 1, 2
bSA3	2, 3	3	n/a	3	n/a	n/a
S10gd/ S13gd	1, 2, 3	1, 2, 3	1, 2, 3	2, 3	2, 3	3
S11gd	1	1, 2	1, 2, 3	2	2, 3	3
S12sd	0	0	0	0, 1	0, 1	0, 1, 2
S13sd	2, 3	3	n/a	3	n/a	n/a
S6sd	k* at: 0, 2 or 3	k* at: 0 or 3	k* at: 0	k* at: 0, 1 or 3	k* at: 0 or 1	k* at: 0, 1 or 2
S5gd	2, 3	3	Para.k at: 0	1, 3	1	2

Table 6.5 Exhaustive test list for the complement of literal (continued)

Comp. of Literal	Test
$\overline{k^1[x]^1}$	sequential: (0, 1), (1, 0), [(1, 2(3)), [2(3), 1] non-sequential: 0, 1, [2(3)]
$\overline{k^1[x]^2}$	sequential: [1(2), 0], [0, 1(2)], [1(2), 3], [3, 1(2)] non-sequential: 0, 3, [1(2)]
$\overline{k^1[x]^3}$	sequential: [0, 1(2)(3)], [1(2)(3), 0] non-sequential: 0, [1(2)(3)]
$\overline{k^2[x]^2}$	sequential: (2, 3), (3, 2), [2, 0(1)], [0(1), 2] non-sequential: 2, 3, [0(1)]
$\overline{k^2[x]^3}$	sequential: [(0, 2(3)) (1, 2(3))], [(2, 0(1)) (3, 0(1))] non-sequential: 1, [2(3)]
$\overline{k^3[x]^3}$	sequential: [0(1)(2), 3], [3, 0(1)(2)] non-sequential: 2, 3

Table 6.6: Complement of literal tests.

Complement of Literal	Test sequence
$\overline{k^1[x]^1}$	$\{(0, 1), (1, 0), (1, 2), (2, 1)\}$ or $\{(0, 1), (1, 0), (1, 3), (3, 1)\}$ Minimal test sequence: $1 \rightarrow 0 \rightarrow 1 \rightarrow 2 \rightarrow 1$ or $1 \rightarrow 0 \rightarrow 1 \rightarrow 3 \rightarrow 1$
$\overline{k^1[x]^2}$	$\{(1, 0), (0, 1), (1, 3), (3, 1)\}$ or $\{(2, 0), (0, 2), (2, 3), (3, 2)\}$ Minimal test sequence: $1 \rightarrow 3 \rightarrow 1 \rightarrow 0 \rightarrow 1$ or $2 \rightarrow 3 \rightarrow 2 \rightarrow 0 \rightarrow 2$
$\overline{k^1[x]^3}$	$\{(1, 0), (0, 1)\}$ or $\{(2, 0), (0, 2)\}$ or $\{(3, 0), (0, 3)\}$ Minimal test sequence: $1 \rightarrow 0 \rightarrow 1$ or $2 \rightarrow 0 \rightarrow 2$ or $3 \rightarrow 0 \rightarrow 3$
$\overline{k^2[x]^2}$	$\{(2, 3), (3, 2), (2, 0), (0, 2)\}$ or $\{(2, 3), (3, 2), (2, 1), (1, 2)\}$ Minimal test sequence: $2 \rightarrow 3 \rightarrow 2 \rightarrow 0 \rightarrow 2$ or $2 \rightarrow 3 \rightarrow 2 \rightarrow 1 \rightarrow 2$
$\overline{k^2[x]^3}$	$\{(1, 2), (2, 1)\}$ or $\{(1, 3), (3, 1)\}$ Minimal test sequence: $1 \rightarrow 2 \rightarrow 1$ or $1 \rightarrow 3 \rightarrow 1$
$\overline{k^3[x]^3}$	$\{(2, 3), (3, 2)\}$ Minimal test sequence: $2 \rightarrow 3 \rightarrow 2$

Table 6.7: Minimal test sets/sequences for the complement of literal.

minimal test sets and sequences are listed in Table 6.10.

6.4 Testing of the *tSum*

Tables 6.11 and 6.12 list the *tSum* tests. The tests in these tables are listed in terms of $Y = x_1 + x_2$. This is possible since all input currents x_1 and x_2 are summed at a common node which is connected to the mirror's input (transistor 1 in Figure 5.17). This is why values greater than 3 appear in the table. The maximum is when $x_1 = 3$ and $x_2 = 3$ making $Y = 6$. When the final results are obtained, they can be expressed in terms of x_1 and x_2 . Table 6.11 lists the tests for the sequential faults. Tests for the sequential* faults are also included. Sequential* faults produce a 3* output when the fault is excited. So, listed valid tests initialize the circuit and then excite the fault with the 3* output. The 3* testing procedure, described later in this chapter, is required in this case. The functional faults tests are listed in Table 6.12. The sum* fault category, which produce 3* output, also require 3* testing. The tests listed excite the 3* output. The SA3* faults, in Table 5.4, can be tested by the same tests of the the SA3 since these test produce a faulty output different from the fault free output. The resultant tests (in terms of $Y = x_1 + x_2$) are tabulated in Table 6.13. As a result, the minimum test set is:

$(Y_1, 1), (1, Y_2)$ or

Fault	X⁻¹	X⁻²	X⁻³
O1g	(0, 1) (0, 2) (0, 3) (1, 2) (1, 3) (2, 1) (2, 3) (3, 1) (3, 2)	(0, 2) (0, 3) (1, 2) (1, 3) (2, 1) (2, 3) (3, 1) (3, 2)	(0, 2) (0, 3) (1, 2) (1, 3) (2, 1) (2, 3) (3, 1) (3, 2)
O1s/O1d	(0, 1) (0, 2) (0, 3) (1, 2) (1, 3) (2, 1) (2, 3) (3, 1) (3, 2)	(0, 2) (0, 3) (1, 0) (1, 2) (1, 3) (2, 0) (2, 3) (3, 0) (3, 2)	(0, 2) (0, 3) (1, 0) (1, 2) (1, 3) (2, 0) (2, 1) (2, 3) (3, 0) (3, 2)
O2g	(0, 1) (0, 2) (0, 3) (1, 0) (2, 0) (3, 0)	(0, 3) (1, 0) (1, 2) (1, 3) (2, 0) (2, 1) (3, 0) (3, 1)	(0, 3) (1, 3) (2, 3) (3, 2)
O3g	(0, 2) (0, 3) (1, 2) (1, 3) (2, 1) (2, 3) (3, 1)	(0, 1) (0, 3) (1, 0) (1, 3) (2, 0) (2, 3) (3, 0) (3, 2)	(0, 1) (0, 2) (1, 0) (1, 2) (2, 0) (2, 1) (3, 0) (3, 1)
O4g	n/a	(0, 2) (0, 3) (1, 2) (1, 3) (2, 1) (3, 1)	(0, 3) (1, 3) (2, 3)
O4s/O4d	(1, 0) (2, 0) (3, 0)	(2, 0) (2, 1) (3, 0) (3, 1)	(3, 0) (3, 1) (3, 2)
O5g	n/a	(1, 3) (2, 0) (3, 0)	(3, 0) (3, 1)
O6g/O7g	(0, 1) (0, 2) (0, 3) (1, 0) (2, 0) (3, 0)	(0, 2) (0, 3) (1, 2) (1, 3) (2, 0) (2, 1) (3, 0) (3, 1)	(0, 3) (1, 3) (2, 3) (3, 0) (3, 1) (3, 2)
O8g	(0, 3)	(0, 3) (1, 3)	n/a
O9g	(0, 2) (0, 3) (1, 2) (1, 3) (2, 3) (3, 2)	(0, 3) (1, 0) (1, 3) (2, 0) (2, 3) (3, 0)	(0, 1) (1, 0) (2, 0) (2, 1) (3, 0) (3, 1)
O10g	(0, 1) (0, 2) (0, 3) (1, 0) (1, 2) (1, 3) (2, 0) (2, 1) (2, 3) (3, 0) (3, 1) (3, 2)	(0, 1) (0, 2) (0, 3) (1, 0) (1, 2) (1, 3) (2, 0) (2, 1) (2, 3) (3, 0) (3, 1) (3, 2)	(0, 1) (0, 2) (0, 3) (1, 0) (1, 2) (1, 3) (2, 0) (2, 1) (2, 3) (3, 0) (3, 1) (3, 2)

Table 6.8: Exhaustive test list for the cycle.

Fault	X⁻¹	X⁻²	X⁻³
O11s/O11d	(1, 0) (2, 0) (3, 0)	(2, 0) (2, 1) (3, 0) (3, 1)	(3, 0) (3, 1) (3, 2)
O12s/O12d	(0, 1) (0, 2) (0, 3)	(0, 2) (0, 3) (1, 2) (1, 3)	(0, 3) (1, 3) (2, 3)
O11g/O12g O13g/O14g	(0, 1) (0, 2) (0, 3) (1, 0) (2, 0) (3, 0)	(0, 2) (0, 3) (1, 2) (1, 3) (2, 0) (2, 1) (3, 0) (3, 1)	(0, 3) (1, 3) (2, 3) (3, 0) (3, 1) (3, 2)
O13s/O13d	(0, 1) (0, 2) (0, 3)	(0, 2) (0, 3) (1, 2) (1, 3)	(0, 3) (1, 3) (2, 3)
O14s/O14d	(1, 0) (2, 0) (3, 0)	(2, 0) (2, 1) (3, 0) (3, 1)	(3, 0) (3, 1) (3, 2)
S2gd	0, 2, 3	0, 2, 3	0, 1, 3
S2sd/S5gs/S6gs/ S7sd/S12sd/S13sd	0	0, 1	0, 1, 2
S3gd	0, 1, 3	1, 2	0, 1, 2, 3
S4gs/S6gd/S11gd/ S13gd	0, 1, 2, 3	0, 1, 2, 3	0, 1, 2, 3
S4gd/S4sd/S7gs/ S11sd/S14sd	1, 2, 3	2, 3	3
S5gd	0, 3	0, 1	2
S5sd	n/a	0	0, 1
S6sd	1, 2, 3	2, 3	n/a
S8gd	1, 2	1, 2	2, 3
S8sd	2, 3	3	n/a
O2S/O2D	1, 2, 3	2, 3	3
O3S/O3D	2, 3	1, 3	1, 2
O5S/O5D	0	0, 1	0, 1, 2
O6S/O6D	0	0, 1	0, 1, 2
O7S/O7D	1, 2, 3	2, 3	0, 1, 3
O8S/O8D	1, 2, 3	2, 3	0, 1, 3
O9S/O9D	1, 2, 3	0, 2, 3	0, 1
SA3	1, 2, 3	0, 2, 3	0, 1, 3
SA0	0, 2, 3	0, 1, 3	0, 1, 2

Table 6.8 Exhaustive test list for the cycle (continued)

Cycle	Test
x^{-1}	sequential: (0, 3), [1(2)(3), 0]
	non-sequential: 0, 2
x^{-2}	sequential: (1, 3), [(2, 0(1)) (3, 0(1))] or [2(3), 0], [0(1), 3]
	non-sequential: 0, 3, [1(2)]
x^{-3}	sequential: [3, 0(1)], [0(1)(2), 3]
	non-sequential: 3, 2, [0(1)]

Table 6.9: Cycle tests.

Cycle	Test sequence
x^{-1}	(0, 3), (2, 0) Minimal test sequence: 2 → 0 → 3
x^{-2}	(1, 3), (3, 0) or (2, 0), (0, 3) Minimal test sequence: 1 → 3 → 0 or 2 → 0 → 3
x^{-3}	(2, 3), (3, 1) or (2, 3), (3, 0) Minimal test sequence: 2 → 3 → 1 or 2 → 3 → 0

Table 6.10: Minimal test sets/sequences for the cycle.

$(Y_1, 2), (2, Y_2)$ where $Y_1, Y_2 \in \{4, 5, 6\}$

The resultant minimal test sequence is:

$Y_1 \rightarrow 1 \rightarrow Y_2$ or

$Y_1 \rightarrow 2 \rightarrow Y_2$

Fault	Tests
O1g	(0, 1) (0, 2) (1, 2) (2, 1) (3, 1) (3, 2) (4, 1) (4, 2) (4, 3) (5, 1) (5, 2) (5, 3) (5, 4) (6, 1) (6, 2) (6, 3) (6, 4)
O2g	(3, 0) (3, 1) (3, 2) (4, 0) (4, 1) (4, 2) (5, 0) (5, 1) (5, 2) (6, 0) (6, 1) (6, 2)
O3g	(0, 1) (0, 2) (1, 0) (1, 2) (2, 0) (2, 1) (3, 0) (3, 1) (3, 2) (4, 0) (4, 1) (4, 2) (5, 0) (5, 1) (5, 2) (6, 0) (6, 1) (6, 2)
O4g	(3, 2) (4, 1) (4, 2) (5, 1) (5, 2) (6, 1) (6, 2)
O5g	(3, 1) (3, 2) (4, 1) (4, 2) (5, 1) (5, 2) (6, 1) (6, 2)
O7g	(0, 3) (0, 4) (0, 5) (0, 6) (1, 3) (1, 4) (1, 5) (1, 6) (2, 3) (2, 4) (2, 5) (2, 6)
O8g	(3, 0) (3, 1) (3, 2) (4, 0) (4, 1) (4, 2) (5, 0) (5, 1) (5, 2) (6, 0) (6, 1) (6, 2)
O8 s/d	(3, 0) (3, 1) (3, 2) (4, 0) (4, 1) (4, 2) (5, 0) (5, 1) (5, 2) (6, 0) (6, 1) (6, 2)
O9g	(3, 0) (3, 1) (3, 2) (4, 0) (4, 1) (4, 2) (5, 0) (5, 1) (5, 2) (6, 0) (6, 1) (6, 2)
O6g*	(0, 3) (0, 4) (0, 5) (0, 6) (1, 3) (1, 4) (1, 5) (1, 6) (2, 3) (2, 4) (2, 5) (2, 6)
O9 s/d*	(0, 3) (0, 4) (0, 5) (0, 6) (1, 3) (1, 4) (1, 5) (1, 6) (2, 3) (2, 4) (2, 5) (2, 6)

Table 6.11: Exhaustive test list for the sequential faults in the $tSum$.

Fault	Tests
S5gs	any Y will test
S6gs	3, 4, 5, 6
S7gs	3, 4, 5, 6
S7gd	any Y will test
O3 s/d	1, 2
O5 s/d	1, 2
O6 s/d	3, 4, 5, 6
O7 s/d	3, 4, 5, 6
Sum*	4, 5, 6
SA3	0, 1, 2
SA0	1, 2, 3, 4, 5, 6

Table 6.12: Exhaustive test list for the functional faults in the *tSum*.

Fault	Test
sequential	[(3, 2)(4, 1(2))(5, 1(2))(6, 1(2))], [(0, 3(4)(5)(6))(1, 3(4)(5)(6))(2, 3(4)(5)(6))]
non-sequential	[1(2)], [4(5)(6)]

Table 6.13: *tSum* tests.

6.5 Testing of the *Min*

A number of points have to be distinguished before listing the *min* tests. The *min* has two inputs each of which is connected to a different node of the circuit, Figure 5.20. Some faults may be specific to a certain node of one of the inputs. This may require that test vectors be specific to that node. For example, the test vector $\langle x, y \rangle = \langle 0, 1 \rangle$ indicates that input $x = 0$ and $y = 1$. x is connected to transistor 1 and y to transistor 2 (Figure 5.20).

When the *min* is used as a block in larger circuits, the physical order of x and y is irrelevant. The logic operation of the *min* is symmetric and there should be no difference between applying $\langle 1, 0 \rangle$ or $\langle 0, 1 \rangle$. If the test vector $\langle 0, 1 \rangle$ works, then $\langle 1, 0 \rangle$ should give the same result. However, as will be seen below, some faults are sensitive to the order in which x and y are applied. To avoid producing test lists with vectors sensitive to the order of x and y , two ways are used during the simulations conducted in this work:

1. If it is found that testing for a certain fault imposes a condition on one node, x or y , the same condition is applied to the other node. This will result in tests independent of the order of applying x or y . For example, if testing a fault requires that node $x \neq 0$ (x connected to transistor 1 in Figure 5.20), the same condition will be applied to y . This forces $y \neq 0$ too (y connected

to transistor 2 in Figure 5.20). This will produce test vectors that are valid regardless of the order.

2. If applying 1, above, is not possible, then both $\langle x, y \rangle$ and $\langle y, x \rangle$ are included in the test list. This will guarantee that the fault is detected and, at the same time, the test list is independent of the order of applying x and y . For example, S4gs can be tested by $\langle 0, 1 \rangle$ in that order (0 is applied to transistor 1's drain and 1 to transistor 2's drain in Figure 5.20). As a result, both $\langle 0, 1 \rangle$ and $\langle 1, 0 \rangle$ are listed in the test list for S4gs. This makes the test list independent of the x, y order.

Another issue is the test generation for sequential faults. Testing such faults requires initializing the circuit by $\langle x_1, y_1 \rangle$ input followed by $\langle x_2, y_2 \rangle$. The number of such possible transitions from $\langle x_1, y_1 \rangle$ to $\langle x_2, y_2 \rangle$ is $16 \times 16 = 256$ (there are $4 \times 4 = 16$ variations of $\langle x_1, y_1 \rangle$ which can be followed by any of the 16 variations of $\langle x_2, y_2 \rangle$). This is a huge number to apply for each fault during the simulations. Instead, the circuit is investigated to find which internal current could actually affect a given transistor. Only transitions of this current are considered and later translated into input pairs of $\langle x, y \rangle$ to find valid tests for faults in that transistor. For a fault in transistor T, for example, only transitions of the gate current I_t , which controls the state of the transistor, are considered ($I_t = 0 \rightarrow 1, 0 \rightarrow 2, 0 \rightarrow 3, 1 \rightarrow 0, 1 \rightarrow 2, \dots$ 12 different transitions). Then, I_t values

are translated into input pairs of $\langle x, y \rangle$ to simulate the fault. Accordingly, the *min* sequential faults simulations are grouped into three different groups.

1. **Sequential part 1:** The gates of transistors 1, 2 and 3 in Figure 5.20 are connected to input x . This suggests considering x transitions only ($0 \rightarrow 1$, $0 \rightarrow 2$, $0 \rightarrow 3$, $1 \rightarrow 0$, ..., 12 different possibilities) when simulating faults in these transistors. The resultant tests are listed in Table 6.14. The test pairs represent $(x1, x2)$ values. We will assign y the same value as x , if possible, to avoid generating order sensitive test vectors. For example, the input test sequence resulting from the pair $(x1, x2) = (0, 1)$ is: $\langle 0, 0 \rangle \rightarrow \langle 1, 1 \rangle$.

Fault	Tests in terms of $(x1, x2)$
O1g	(0, 1) (0, 2) (2, 1) (3, 1) (3, 2)
O1s/d	(0, 1) (0, 2) (1, 0) (1, 2) (2, 0) (2, 1) (3, 0) (3, 1) (3, 2)
O2g	(3, 1) (3, 2)
O3g	(0, 1) (0, 2) (0, 3) (1, 0) (1, 2) (1, 3) (2, 0) (2, 1) (2, 3) (3, 0) (3, 1) (3, 2)

Table 6.14: Exhaustive test list for the *min*, sequential part 1.

2. **Sequential part 2:** The gates of transistors 6 and 7 are connected to I4 (Figure 5.20). I4 transitions will be used ($I4 = \min(x, y)$). In this case, values of y can also be the same as x to avoid x, y order dependant tests. Table 6.15 lists the resultant tests in terms of $(x1, x2)$ for this category.
3. **Sequential part 3:** Faults O5g and O6g depend on $I2 = x - y$ (Figure 5.20).

Fault	Tests in terms of (x1, x2)
O6g	(0, 1) (0, 2) (1, 2) (2, 1) (3, 1) (3, 2)
O6s/d	(0, 1) (0, 2) (1, 0) (1, 2) (1, 3) (2, 0) (2, 1) (3, 0) (3, 1)
O7g	(0, 1) (0, 2) (0, 3) (1, 0) (1, 2) (1, 3) (2, 0) (2, 1) (2, 3) (3, 0) (3, 1) (3, 2)

Table 6.15: Exhaustive test list for the *min*, sequential part 2.

So, I2 transitions were used to find the tests. Table 6.16 lists the resultant tests in terms of I2 values. It should be noted that the minimum list for this category is the O4g tests since they are all included in the list of O5g. To translate I2 values into input sequences of $\langle x1, y1 \rangle \rightarrow \langle x2, y2 \rangle$, Table 6.17 is used. $\langle 0, y \rangle$, $\langle x, 0 \rangle$ and $\langle 3, 3 \rangle$ pairs will be excluded to avoid making the faulty output equal to the correct output in the presence of SA0 or SA3 faults. In the case of $I2 = 3$, $\langle 3, 0 \rangle$ was the only choice since no other value of x and y satisfy $3 = x - y$. The resultant tests in terms of x and y for each I2 pair are listed in Table 6.18. It should be noted that these tests (in Table 6.18) are sensitive to the order in which x and y are applied and it is not possible to assign x and y the same values. So, to avoid order sensitive test lists, as described earlier, both of $\langle x, y \rangle$ and $\langle y, x \rangle$ should be added to the test list. So, each $\{\langle x1, y1 \rangle, \langle x2, y2 \rangle\}$ in Table 6.18 have to be applied as follows:

1. $\langle x1, y1 \rangle \rightarrow \langle x2, y2 \rangle$ and
2. $\langle y1, x1 \rangle \rightarrow \langle y2, x2 \rangle$

This will resolve the dependency on x, y order. For example, if the pairs $\{ \langle 1, 3 \rangle, \langle 2, 1 \rangle \}$ are added to the test set, the pairs $\{ \langle 3, 1 \rangle, \langle 1, 2 \rangle \}$ have to be added too. The resultant test sequences from these two pairs are $\langle 1, 3 \rangle \rightarrow \langle 2, 1 \rangle$ and $\langle 3, 1 \rangle \rightarrow \langle 1, 2 \rangle$

Fault	Tests in terms of $(I2_1, I2_2)$
O4g	(0, 1) (0, 2) (1, 2) (2, 1) (3, 1) (3, 2)
O5g	(0, 1) (0, 2) (0, 3) (1, 0) (1, 2) (1, 3) (2, 0) (2, 1) (2, 3) (3, 0) (3, 1) (3, 2)

Table 6.16: Exhaustive test list (in terms of I2) for the *min*, sequential part 3.

I2	Corresponding $\langle x, y \rangle$ input (order is significant)
0	$\langle 0, 0 \rangle, \langle 0, 1 \rangle, \langle 0, 2 \rangle, \langle 0, 3 \rangle$ $\langle 1, 1 \rangle, \langle 1, 2 \rangle, \langle 1, 3 \rangle$ $\langle 2, 2 \rangle, \langle 2, 3 \rangle, \langle 3, 3 \rangle$
1	$\langle 1, 0 \rangle, \langle 2, 1 \rangle, \langle 3, 2 \rangle$
2	$\langle 2, 0 \rangle, \langle 3, 1 \rangle$
3	$\langle 3, 0 \rangle$

Table 6.17: I2 values and the corresponding $\langle x, y \rangle$ input values.

For the functional faults (SA0 and SA3 are included), two categories are found: faults that can be tested by vectors independent of the order in which x and y are applied and faults sensitive to the order of x and y . Table 6.19 lists the simplified test list for faults independent of the x, y order. Any $\langle x, y \rangle$ test input in this table can be applied as either $\langle x, y \rangle$ or $\langle y, x \rangle$. Table 6.20 lists the tests for the x, y order sensitive faults. To make the list independent of the x, y order, both

I2 pair	Tests in terms of $\{ \langle x1, y1 \rangle, \langle x2, y2 \rangle \}$ (x, y order is significant)
(0, 1)	$\{ \langle 1, 1 \rangle, \langle 2, 1 \rangle \}, \{ \langle 1, 1 \rangle, \langle 3, 2 \rangle \}, \{ \langle 1, 2 \rangle, \langle 2, 1 \rangle \},$ $\{ \langle 1, 2 \rangle, \langle 3, 2 \rangle \}, \{ \langle 1, 3 \rangle, \langle 2, 1 \rangle \}, \{ \langle 1, 3 \rangle, \langle 3, 2 \rangle \},$ $\{ \langle 2, 2 \rangle, \langle 2, 1 \rangle \}, \{ \langle 2, 2 \rangle, \langle 3, 2 \rangle \}, \{ \langle 2, 3 \rangle, \langle 2, 1 \rangle \},$ $\{ \langle 2, 3 \rangle, \langle 3, 2 \rangle \}$
(0, 2)	$\{ \langle 1, 1 \rangle, \langle 3, 1 \rangle \}, \{ \langle 1, 2 \rangle, \langle 3, 1 \rangle \}, \{ \langle 1, 3 \rangle, \langle 3, 1 \rangle \},$ $\{ \langle 2, 2 \rangle, \langle 3, 1 \rangle \}, \{ \langle 2, 3 \rangle, \langle 3, 1 \rangle \}$
(1, 2)	$\{ \langle 2, 1 \rangle, \langle 3, 1 \rangle \}, \{ \langle 3, 2 \rangle, \langle 3, 1 \rangle \}$
(2, 1)	$\{ \langle 3, 1 \rangle, \langle 2, 1 \rangle \}, \{ \langle 3, 1 \rangle, \langle 3, 2 \rangle \}$
(3, 1)	$\{ \langle 3, 0 \rangle, \langle 2, 1 \rangle \}, \{ \langle 3, 0 \rangle, \langle 3, 2 \rangle \}$
(3, 2)	$\{ \langle 3, 0 \rangle, \langle 3, 1 \rangle \}$

Table 6.18: Exhaustive test list for the *min*, sequential part 3.

$\langle x, y \rangle$ and $\langle y, x \rangle$ tests are required. For example, if $\langle 0, 1 \rangle$ is selected to test for O2d, $\langle 1, 0 \rangle$ have to be applied too.

Fault	Tests in terms of $\langle x, y \rangle$ (order is not important)
S3gd	$\langle 0, 0 \rangle \langle 0, 1 \rangle \langle 0, 2 \rangle \langle 0, 3 \rangle$ $\langle 1, 1 \rangle \langle 1, 2 \rangle \langle 1, 3 \rangle \langle 2, 2 \rangle$ $\langle 2, 3 \rangle$
S2gd	$\langle 3, 3 \rangle \langle 3, 2 \rangle \langle 2, 1 \rangle \langle 2, 2 \rangle$ $\langle 1, 1 \rangle$
S5gd	$\langle 3, 1 \rangle \langle 3, 3 \rangle$

Table 6.19: Exhaustive test list for the *min*, functional part 1.

The above procedure reduced the time needed to find valid tests. There may be other tests that were not investigated by considering only certain node currents in finding tests for the sequential faults. However, the procedure is supported by some selective simulations and found to cover all possible tests (further proof or exhaustive simulation to prove this has not been done). Fortunately, the resultant

Faults	Tests in terms of $\langle x, y \rangle$ (order is significant)
S4gs, S4sd, S5gs, O2s and O2d	$\langle 0, 1 \rangle$ & $\langle 1, 0 \rangle$ $\langle 0, 2 \rangle$ & $\langle 2, 0 \rangle$ $\langle 0, 3 \rangle$ & $\langle 3, 0 \rangle$ $\langle 1, 2 \rangle$ & $\langle 2, 1 \rangle$ $\langle 1, 3 \rangle$ & $\langle 3, 1 \rangle$ $\langle 2, 3 \rangle$ & $\langle 3, 2 \rangle$

Table 6.20: Exhaustive test list for the *min*, functional part 2.

tests produced short minimal sequences that test for all faults. The minimal test sequence is obtained by first combining and simplifying the tests for the sequential parts 1/2 (Tables 6.14 and 6.15). The result will also test for the functional part 1 (Table 6.19). The minimal test sequence for the *min* is then found by combining with the tests for the sequential part 3 (Table 6.18) and functional part 2 (Table 6.20). The detailed steps and the minimal test sequences for the *min* are listed in Table 6.21. The table also lists longer test sequences to avoid the transition $\langle 3, 3 \rangle \rightarrow \langle 1, 1 \rangle$ which requires both inputs to jump from 3 to 1 at the same time.

It should be noted that the issue of test invalidation is not considered in this work. For example, when the sequence $\langle 3, 3 \rangle \rightarrow \langle 2, 2 \rangle$ is applied it could pass through intermediate stages like $\langle 3, 3 \rangle \rightarrow \langle 3, 2 \rangle \rightarrow \langle 2, 2 \rangle$ which may, or may not, invalidate the test.

Step	Test sequence
1	from sequential parts 1 and 2: minimal (x1, x2) set is {(3, 1)} the corresponding test sequence is: < 3,3 > → < 1, 1 >
2	1 (above) will test for the functional part 1
3	combining with sequential part 3 and functional part 2, the minimal test sequences for the <i>min</i> : < 3,3 > → < 1, 1 > → < 2, 1 > → < 1, 1 > → < 1, 2 > or < 3,3 > → < 1, 1 > → < 3, 2 > → < 1, 1 > → < 2, 3 > or < 3,3 > → < 1, 1 > → < 3, 1 > → < 1, 1 > → < 1, 3 >
4	to avoid the transition < 3,3 > → < 1, 1 >, longer test sequences are used: < 3,3 > → < 2, 2 > → < 2, 1 > → < 2, 2 > → < 1, 2 > → < 2, 2 > → < 1, 1 > or < 3,3 > → < 2, 2 > → < 2, 1 > → < 2, 2 > → < 1, 2 > → < 1, 1 > → < 2, 2 > or < 3,3 > → < 2, 2 > → < 3, 2 > → < 2, 2 > → < 2, 3 > → < 2, 2 > → < 1, 1 >

Table 6.21: Minimal test sets/sequences for the *min*

6.6 Peculiar Faults Testing and Fault Coverage

Some faults result in unusual or unstable faulty behavior. One fault in three circuits (literal, complement of literal and *tSum*) results in oscillations at the output. One fault in the *tSum* circuit reduces the logic 3 current (normally 60 uA) to 50 uA. The 3* fault, defined earlier, results in current levels higher than 60 uA (the fault free maximum value). Description of these faults along with a testing procedure for 3* faults are presented below.

6.6.1 Oscillations Fault

Under this fault, the circuit's output is not stable. It oscillates over the whole range from 0 to 60uA with high frequency. This fault is not covered by the test sequences derived earlier. However, only one short produced this type of fault (S6gd in the literal and complement of literal, and S3gd in the *tSum*) which is a low percentage (1.3% for the literals and 1.9% for the *tSum*).

In the literal and complement of literal circuits, this fault is caused by the continuous charging / discharging paths affecting the gate voltage (N_{ref}) of transistors 4, 5 and 6 in Figures 5.2 and 5.9. The charging path sources from the output current (when the switch transistor 7 is open) through the short to the gate of transistor 6 (which is connected to the gates of 4 and 5). The discharging path (when transistor 7 is closed) passes through the short to the grounded source of transistor 6. Figure 6.1 illustrates the charging and discharging paths in the literal circuit.

In the *tSum* circuit, this fault is caused by the continuous charging / discharging paths affecting the gate voltage of transistors 1, 2 and 3 in Figure 5.17. The charging path sources from v_{dd} through transistor 3 (when conducting) to its gate through the short. This will charge the gate and will close transistors 1, 2 and 3. As a result, no current will flow through transistor 2 and node A voltage will be binary 0 making node b voltage binary 1. This in turn will open transistor 5 which activates

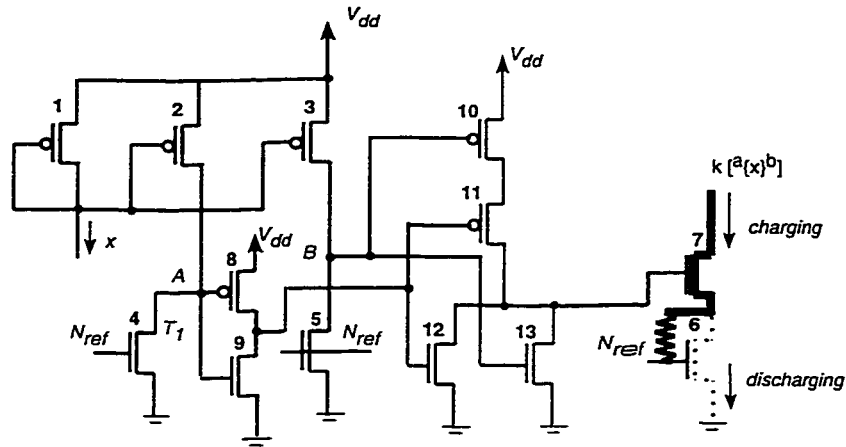


Figure 6.1: Fault S6gd charging and discharging paths (the literal circuit).

the discharging path and transistors 1, 2 and 3 will conduct again. Now node A will be binary 1 and node B will be binary 0 which closes transistor 5 allowing the charging path to be activated again and the cycle will repeat continuously. Figure 6.2 illustrates the charging and discharging paths in the $tSum$ circuit.

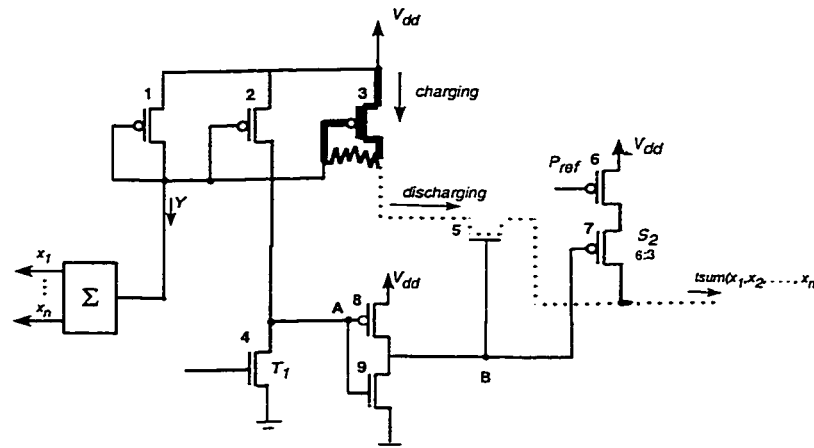


Figure 6.2: Fault S3gd charging and discharging paths (the $tSum$ circuit).

This can be solved (verified by experiment) for the literal and complement of literal if the positions of transistors 6 and 7 (see Figures 5.2 and 5.9) are swapped (a Design for Testability (DFT) issue). In this case, the fault will be converted to a normal fault that can be detected by the test sequences. No similar solution was found for the *tSum*.

6.6.2 A Soft Error

This soft fault slightly changes the output into values which may, or may not, be different from the fault free output. One short, S6gd in the *tSum*, resulted in a slightly different output from the fault free circuit. It only changes the current value of logic 3 (normally 60uA) into an intermediate level (50uA) between 2 and 3. This current level can be read as 3, and passes undetected, or as 2 which can be detected. This fault is considered uncovered by the test sequences produced.

6.6.3 3* Faults

As mentioned earlier, the normal currents of the logical levels in the circuits studied are: 0 for logic 0, 20uA for logic 1, 40uA for logic 2 and 60uA for logic 3. 3* faults refer to any fault that results in current levels higher than 60uA. Typical values found in the simulations range from 80uA to 300uA.

If a 3^* value is produced at the output of a faulty circuit, it may act as an input to another circuit. If the fault free value for a 3^* fault is 0, 1 or 2, then the fault will be detected. However, when the fault free value is 3, then the behavior of each circuit needs to be studied to find out whether this fault will produce an observable effect on the output or not. The results of this study are summarized below.

1. Literal, complement of literal and *tSum* gates will read the 3^* value as 3 and their outputs will not be affected. This is due to the current threshold elements at the inputs of these circuits which read any value $\geq 60\mu\text{A}$ as 3.
2. The cycle will always produce different outputs for 3 and 3^* inputs. Therefore, the fault can be detected. The cycle output is determined by two main currents: the input current, x , (mirrored through transistor 3 in Figure 5.13) and a constant added (transistor 5 in Figure 5.13) to x or subtracted (transistor 8 in Figure 5.13) from x . So, the actual input current value will always participate in generating the output and will produce a distinguishable difference if it is higher than 60 μA .
3. The *min* will respond to a 3^* fault in two different ways. First, if one input is 3^* and the other input $\neq 0$, it will read 3^* as if it was normal 3. On the other hand, if the input was $\langle 3^*, 0 \rangle$ (or $\langle 0, 3^* \rangle$) the output will be a recognized logic level higher than 1 and the fault will be detected. This can be explained by the *min* implementation which relies on accumulative

subtraction and addition operations performed on the input currents. The accumulative error will be higher if one of the inputs is 0. This was also verified by simulation.

In the previously generated test sequences, the following should be noted:

1. The 3^* fault can be detected if there is a fault free output different from 3. For example, $SA3^*$ fault in the cycle can be detected by tests that produce an output that is different than 3. For the cycle X^{-1} the fault free output is 0 when the input 1 is applied. This fault free output is distinguishable from the 3^* faulty output and the fault is detected. If a 3^* fault is detectable, the test sequence will generate a test for it.
2. The 3^* fault can not be detected directly by a test input if the fault free output is 3. In this case, the faulty output is 3^* and the fault free output is 3. For example, the literal $3^1[x]^2$ fault free output is 3 when the input 1 is applied. In the presence of a 3^* fault, the faulty output will be 3^* . In this case, the input value of 1 is not a test for the fault but it excites the 3^* behavior and further steps (3^* testing procedure below) need to be applied. If the 3^* fault can not be detected, the test sequence is designed to excite the fault.

The circuits that produce un-detectable 3^* faults are: literal, complement of

literal, and $tSum$. For the literals, if the k is 1 or 2 and a 3^* output is produced, it will be detected since it will produce different logic level, 3. For literals when k is set to 3 and the $tSum$, special propagation procedure needs to be taken when the correct output is 3 and the faulty output is 3^* (this happens when x in $[a, b]$ for the literal, x outside $[a, b]$ for the complement of literal, and $x_1 + x_2 \geq 3$ for the $tSum$). The recommended procedure is shown below.

1. If the subsequent circuit, whose input is connected to the 3^* , is literal, complement of literal or $tSum$, it will not be affected and will produce correct values. In this case, the error will be redundant.
2. If the subsequent circuit is a cycle, it will always produce faulty output which can then be propagated to an observable output.
3. If the subsequent circuit is a min , then the test algorithm should propagate the expected 3^* twice. Once with the other min 's input set to 0 and once when it is set to 3. The 0 will detect a 3^* fault and the 3 will detect other faults.

3* Effect on the Fault Coverage

The fault coverage figures for the generated test sequences in this work are tabulated in Table 6.22. Two figures are given, one if the 3^* procedure is applied and the other

if the procedure is not applied. The first assumes that the above procedure of 3* testing is applicable. The second is when the 3* procedure is not applied.

Circuit	Un-covered faults	Coverage, 3* test applicable	# of un-detectable 3* faults	Coverage, 3* test not applicable
Literal	S6gd (Osc.)	98.7%	2 (S6sd, O6g)	98.7 - (2.6 * α) 98.7% if $\alpha = 0$ 96.1% if $\alpha = 1$
Comp. of literal	S6gd (Osc.)	98.7%	2 (S6sd, O6g)	98.7 - (2.6 * α) 98.7% if $\alpha = 0$ 96.1% if $\alpha = 1$
Cycle	0	100%	0	100%
tSum	S6gd (Osc.) S3gd (Para.)	96.2%	11 (sum* + sequential*)	75.4%
Min	0	100%	0	100%

Table 6.22: Fault coverage figures and 3* testing.

α , in Table 6.22 is the ratio of the number of literal / complement of literal gates that are designed with $k = 3$ compared to the number of literal / complement of literal gates in the whole network. It appears from the above that the 3* fault is more serious in the *tSum* gate. For this, a number of recommendations for testability will be introduced and discussed in the following chapter.

Chapter 7

Design for Testability

Based on the results obtained in this work, the following general recommendations for Design for Testability (DFT) can be made for the set of MVL circuits considered:

1. Literals and *tSum* should not be used as output stages. This is to avoid 3* values at the output where its effect can propagate to another circuit. In a typical MVL circuit, literals appear at the input stages and do not appear afterwards.
2. If a *tSum* has to be used at the output, it should be followed by 2 consecutive cycle gates. This will guarantee the detection of the 3* faults if they appear at the *tSum* output.

3. Apply 3* test procedure for every *min* preceded by literals or *tSum*. This is done in order to properly propagate possible 3* values generated by the literals or the *tSum*.
4. For the Literal and Complement of Literal gates, swap the positions of transistors 6 and 7. This will eliminate the oscillation fault caused by the S6gd short and will convert it to a testable fault.

From the above, the following general statement can be made:

MVL sets that include *tSum* gates should be avoided. This is because the *tSum* requires to be followed by 2 Cycle gates, to avoid passing 3* faults. However, to validate this statement, two costs have to be compared. First, using MVL sets that do not include the *tSum* may increase the number of operators (and hence the number of circuits) required to implement a given function. This cost needs to be checked against the cost of having *tSums* followed by Cycles.

7.1 An Application

After using the testability recommendations in this work, the given test sequences can be used in testing larger circuits. There are already some approaches which use hierarchical testing with pre-computed test sets for the basic blocks in the network.

The work of Sarfert et. al. [33], and Calhoun and Breglez [3] are typical examples. In this method, pre-computed test sets for the basic blocks of larger networks, are used during testing. Faults on the other lines in the network are propagated normally. When a block is to be tested, its pre-computed test set inputs are propagated to the block inputs through the network. Then, its output is propagated to final network outputs.

For example, this concept can be applied to an MVL full adder. The truth table of the adder is given in Table 7.1. The inputs are a_i , b_i and the previous stage carry C_{i-1} . The output sum is S_i with the carry C_i . The corresponding circuit shown in Figure 7.1. Two *tSum* operators are found in the circuit. As recommended above, each *tSum* circuit has to be followed by two cycle circuits. Another solution is to design the full adder using sets that do not contain the *tSum* operator. The full adder circuit is modified by adding cycles after each *tSum* and presented in Figure 7.2. The inserted cycle circuits are numbered as DFT1 to DFT4.

The following sequence of a_i b_i will test for all faults in gates 1 to 7:

$$a_i = 1, 0, 1, 2, 1, 2, 3, 2, 0, 2, 1, 1, 1, 2, 2, 2, 3, 3, 3, 0, 1, 2, 3$$

$$b_i = 2, 3, 2, 1, 2, 1, 0, 1, 3, 1, 2, 3, 1, 1, 3, 0, 2, 0, 3, 0, 3, 2, 1.$$

Similarly, the sequences can be extended to test the rest of the circuit. It should be noted that it may not be possible to propagate the minimal test sequence for

a given gate inside the circuit. In this case, other longer test sequences should be used (all test are pre-computed earlier) and if still not possible to propagate, then the gate will be un-testable for some faults.

$a_i b_i$	C_{i-1}	S_i	C_i
0 0	0/1	0/1	0/0
0 1	0/1	1/2	0/0
0 2	0/1	2/3	0/0
0 3	0/1	3/0	0/1
1 0	0/1	1/2	0/0
1 1	0/1	2/3	0/0
1 2	0/1	3/0	0/1
1 3	0/1	0/1	1/1
2 0	0/1	2/3	0/0
2 1	0/1	3/0	0/1
2 2	0/1	0/1	1/1
2 3	0/1	1/2	1/1
3 0	0/1	3/0	0/1
3 1	0/1	0/1	1/1
3 2	0/1	1/2	1/1
3 3	0/1	2/3	1/1

Table 7.1: Truth table for an MVL full adder.

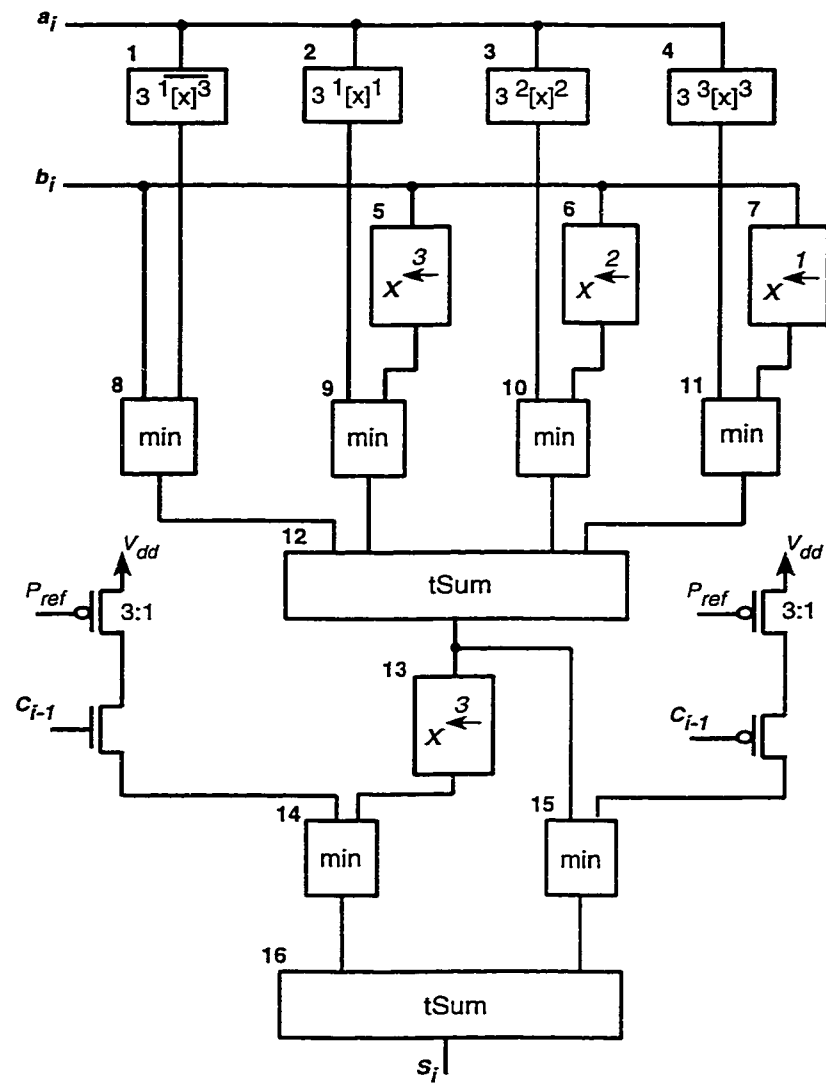


Figure 7.1: A 4-valued adder.

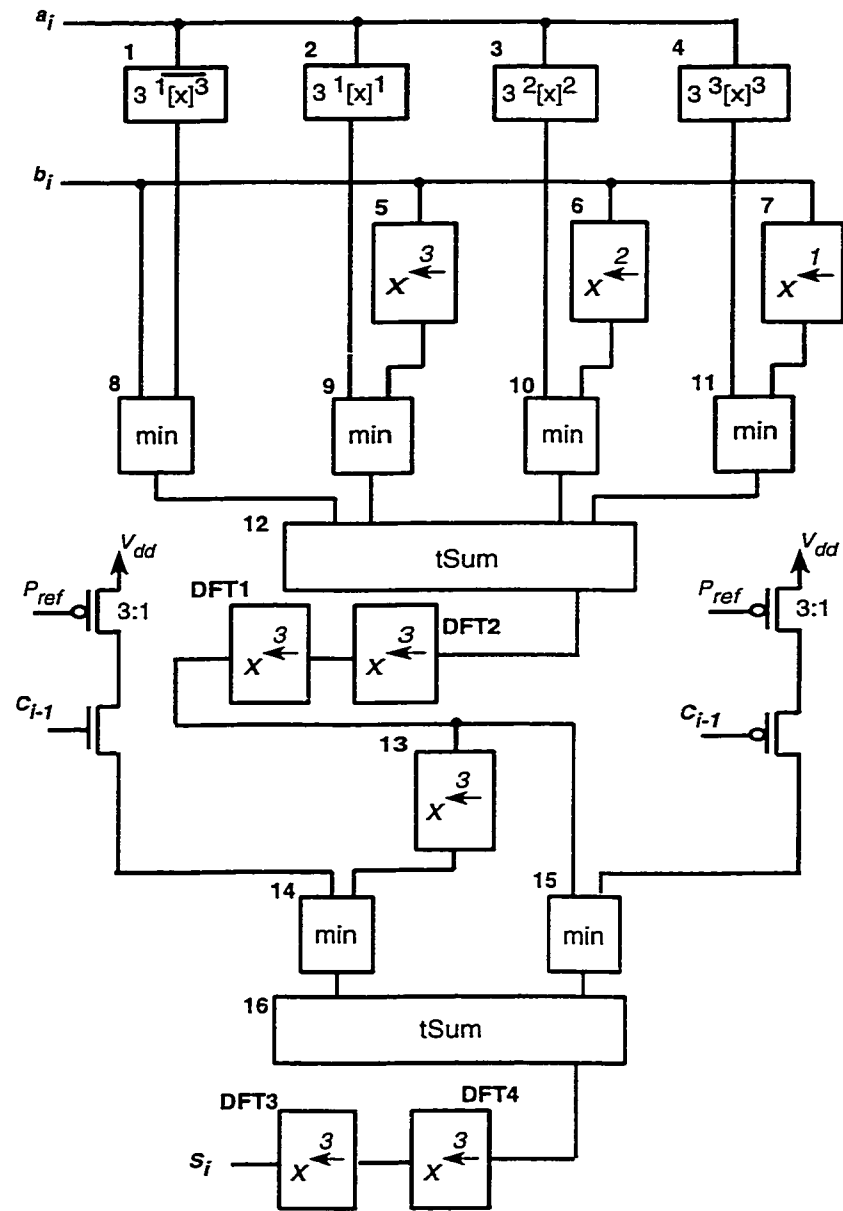


Figure 7.2: A testable 4-valued adder.

Chapter 8

Discussion and Conclusions

In this work, fault characterization and testability considerations in a set of MVL basic gates were considered. To achieve this, faults (shorts and opens) were inserted one at a time and each circuit was simulated for all possible input transitions to determine which inputs are valid to test for the fault inserted. The process was repeated for each circuit for all its faults and the resultant test vectors for all faults were tabulated. These tables were then used to generate minimal test sequences for each circuit. These sequences are derived by using the fault coverage table method. They form along with the whole tests lists an essential input for any test procedure for MVL networks built using the considered primitives primitives.

8.1 Summary of the Main Results of the Thesis

The fault characterization conducted on the MVL set in this work resulted in 4 main fault categories:

1. **Sequential faults:** which have sequential behavior.
2. **SA0:** the output is stuck-at zero level.
3. **SA3:** the output is stuck-at level 3. In the case of literals, this appeared as stuck at the literal value (the value of k).
4. **Functional faults:** which change the circuit function into a different one from what it was designed for.

In addition, some other peculiar faults were found with low percentages. The percentages of each category against the circuits studied are summarized in Table 8.1.

Fault category	Literal	Comp. of Literal	Cycle	tSum	Min
Sequential	33.8%	33.8%	31.7%	24.6%	28.2%
Functional	20.8%	20.8%	53.7%	37.7%	30.8%
SA0	40.3%	6.5%	6.1%	7.5%	33.3%
SA3	1.3%	35.1%	8.5%	26.4%	7.7%
Others	3.9%	3.9%		3.8%	

Table 8.1: Summary of the characterized fault categories.

The main conclusions that can be derived from this work are:

1. The stuck-at fault model is inadequate for representing faults in the MVL circuits. Actually, it represents less than 40% of the total faults.
2. High percentage of faults appeared as functional faults, which change the circuit output into different function. They ranged from 20% in the literals 53.7% in the cycle.
3. Only SA0 or SA3 faults appeared. There are no stuck-at faults at the other logic levels 1 or 2 (stuck-at the literal value, in the literals, is a special case of SA3).

8.2 Future Work

The use of the fault categories and pre-computed tests, found in this work, is a subject for future work. Fault categories found can be used to define realistic fault models to be used by ATPGs. The pre-computed test sets can be used in hierarchical testing where small blocks in larger networks are tested using their pre-computed tests. Also, as a result of the low testability of the *tSum* circuit, cost comparison studies need to be conducted. The cost of using MVL sets that do not contain the *tSum* operator needs to be compared against those sets that use *tSums* where each *tSum* is followed by two extra cycle circuits to improve testability. Other similar fault characterization studies on other MVL implementations are needed since this

area of research has remained un-attended. The fault models, being used in MVL testing, are heavily based on the stuck-at and skew fault models. Other fault models should be subject to future work.

Bibliography

- [1] Y. M. Ajabnoor and Mostafa H. Abd-El-Barr. Stuck-type Fault Detection in MVL Combinational Circuits. In *Proceedings - The Eleventh International Symposium on Multi-valued Logic (ISMVL)*, pages 275–282, 1981.
- [2] N. Burgess, R. I. Damper, K. A. Totton, and S. J. Shaw. Physical Faults in CMOS Circuits and Their Coverage by Different Fault Models. *IEE Proceedings-E*, 135(1):1–9, 1988.
- [3] John D. Calhoun and Franc Brglez. A Framework and Method for Hierarchical Test Generation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(1):45–67, 1992.
- [4] Brain Chess, Anthony Freitas, F. Joel Ferguson, and Tracy Larrabee. Testing CMOS Logic Gates for Realistic Shorts. In *International Test Conference*, pages 395–402, 1994.
- [5] Rob Dekker, Frans Beenker, and Loek Thijssen. A Realistic Fault Model and Test Algorithms for Static Random Access Memories. *IEEE Transactions on Computer-Aided Design*, 9(6):567–572, 1990.
- [6] Rolf Drechsler, Martin Keim, and Bernd Becker. Fault Simulation in Sequential Multi-Valued Logic Networks. In *Proceedings of the Twenty-Seventh International Symposium on Multiple-Valued Logic*, pages 145–150, May 1997.
- [7] Rolf Drechsler, Rolf Krieger, and Bernd Becker. Random Pattern Fault Simulation in Multi-Valued Circuit. In *Proceedings of the Twenty-Fifth International Symposium on Multiple-Valued Logic*, pages 98–103, May 1995.
- [8] E.V. Dubrova, D.B. Gurov, and J.C. Muzio. Full Sensitivity and Test Generation for Multiple-Valued Logic Circuits. In *Proceedings of the Twenty-Fourth International Symposium on Multiple-Valued Logic*, pages 284–288, May 1994.

- [9] E.V. Dubrova, D.B. Gurov, and J.C. Muzio. The Evaluation of Full Sensitivity for Test Generation in MVL Circuits. In *Proceedings of the Twenty-Fifth International Symposium on Multiple-Valued Logic*, pages 104–109, May 1995.
- [10] M. Favalli, P. Olivo, M. Damiani, and B. Ricco. Fault Simulation of Unconventional Faults in CMOS Circuits. *IEEE Transactions of Computer-Aided Design*, 10(5):677–682, May 1991.
- [11] H. Fujiwara and Shimono T. On the Acceleration of Test Generation Algorithm. *IEEE Transactions on Computers*, December:1137–1155, 1983.
- [12] J. Galiary, Y. Crouzet, and M. Vergniault. Physical Versus Logical Fault Models MOS LSI Circuits: Impact on Their Testability. *IEEE Transactions on Computers*, C-29(6):529–531, June 1980.
- [13] P. Goel. An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits. *IEEE Transactions on Computers*, March:215–222, 1981.
- [14] Hong Hao and Edward J. McCluskey. On the Modeling and Testing of Gate Oxide Shorts in CMOS Gates. In *International Workshop on Defect and Fault Tolerance on VLSI Systems*, pages 161–174, 1991.
- [15] Hong Hao and Edward J. McCluskey. Analysis of Gate Oxide Shorts in CMOS Circuits. *IEEE Transactions on Computers*, 42(12):1510–1516, December 1993.
- [16] Charles F. Hawkins and Jerry M. Soden. Electrical Characteristics and Testing Considerations for Gate Oxide Shorts in CMOS ICs. In *International Test Conference*, pages 544–555, 1985.
- [17] S. Hessabi, M. Y. Osman, and M. I. Elmasry. Differential BiCMOS Logic Circuits: Fault Characterization and Design-for-Testability. *IEEE Transactions on VLSI Systems*, 3(3):437–445, September 1995.
- [18] Mou Hu. Design of One-Vector Testable Binary Systems Based on Ternary Logic. In *Proceedings of the Twenty-sixth International Symposium on Multiple-Valued Logic*, pages 62–66, May 1996.
- [19] Marcel Jacomet and Walter Guggenbuhl. Layout-Dependent Fault Analysis and Test Synthesis for CMOS Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(6):888–899, 1993.
- [20] Atul K. Jain, Ron J. Bolton, and Mostafa H. Abd-El-Barr. CMOS Multiple-Valued Logic Design-Part I: Circuit Implementation. *IEEE Transactions on Circuits and Systems-I: Fundamental Theory and Applications*, 40(8):503–514, 1993.

- [21] Atul K. Jain, Ron J. Bolton, and Mostafa H. Abd-El-Barr. CMOS Multiple-Valued Logic Design-Part II: Function Realization. *IEEE Transactions on Circuits and Systems-I: Fundamental Theory and Applications*, 40(8):515–522, 1993.
- [22] M. Kameyama, S. Kawahito, and T. Higuchi. A Multiplier Chip and Multiple-valued Bidirectional Current Mode Logic Circuits. *IEEE Computer*, 21:43–56, April 1988.
- [23] Martin Keim, Nicloe Drechsler, Rolf Drechsler, and Bernd Becker. Test Generation for (Sequential) Multi-Valued Logic Networks based on Genetic Algorithm. In *Proceedings - The Twenty-Eighth International Symposium on Multi-valued Logic (ISMVL)*, pages 215–220, 1998.
- [24] Marc E. Levitt, Kaushik Roy, and Jacob A. Abraham. BiCMOS Fault Models: Is Stuck-At Adequate? In *IEEE International Conference on Computer Design*, pages 294–297, 1990.
- [25] Siyad C. Ma and Edward J. McCluskey. Non-Conventional Faults in BiCMOS Digital Circuits. In *International Test Conference*, pages 882–891, 1992.
- [26] Wojciech Maly. Modeling of Lithography Related Yield Losses for CAD of VLSI Circuits. *IEEE Transactions on Computer-Aided Design*, 4(3):166–177, 1985.
- [27] Mohamed Y. Osman and Mohamed I. Elmasry. Highly Testable Design of BiCMOS Logic Circuits. *IEEE Journal of Solid-State Circuits*, 29(6):671–678, 1994.
- [28] D. A. Rich. A Survey of Multivalued Memories. *IEEE Transactions on Computers*, C-35:99–106, February 1986.
- [29] R. Rodriguez-Montanes, E. M. J. G. Bruls, and J. Figueras. Bridging Defects Resistance Measurements in a CMOS Process. In *International Test Conference*, pages 892–899, 1992.
- [30] J. P. Roth. Diagnosis of automata failures: a calculus and a method. *IBM Journal of Research and Development*, 10:278–291, 1966.
- [31] Kaushik Roy, Marc E. Levitt, and Jacob A. Abraham. Testing Considerations for BiCMOS Logic Families. In *Custom Integrated Circuits Conference*, pages 17.2.1–17.2.4, 1991.
- [32] Aly E. Salama and Mohamed I. Elmasry. Fault Characterization, Testing Considerations, and Design for Testability of BiCMOS Logic Circuits. *IEEE Journal of Solid-State Circuits*, 27(6):944–947, June 1992.

- [33] Thomas M. Sarfert, Remo G. Markgraf, Michael H. Schulz, and Edwin Trischler. A Hierarchical Test Pattern Generation System Based on High-Level Primitives. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(1):34–44, 1992.
- [34] J. Segura, A. Rubio, and J. Figueras. Analysis and Modeling of MOS Devices With Gate Oxide Short Failures. In *Proceedings - IEEE International Symposium on Circuits and Systems*, pages 2164–2167, 1991.
- [35] John P. Shen, W. Maly, and F. Joel Ferguson. Inductive Fault Analysis of MOS Integrated Circuits. *IEEE Design and Test of Computers*, 2(6):13–26, 1985.
- [36] Vlad P. Shmerko, V. Levashenko, and I. Bondar. Technique of Computing Logic Derivatives for MVL-Functions. In *Proceedings of the Twenty-Sixth International Symposium on Multiple-Valued Logic*, pages 267–272, May 1996.
- [37] Vlad P. Shmerko, S. Yanushkevich, and V. Levashenko. Test Pattern Generation for Combinational Multi-Valued Networks based on Generalized D-algorithm. In *Proceedings of the Twenty-Seventh International Symposium on Multiple-Valued Logic*, pages 139–144, May 1997.
- [38] Jerry M. Soden and Charles F. Hawkins. Test Considerations for Gate Oxide Shorts in CMOS ICs. In *IEEE Design and Test of Computers*, pages 56–64, August 1986.
- [39] Richard J. Spillman and Stephen Y.H. Su. Detection of Single, Stuck-Type Failures in Multivalued Combinational Networks. *IEEE Transactions on Computers*, C-26(12):1242–1251, December 1977.
- [40] C. H. Stapper. Modeling of Integrated Circuit Defect Sensitivities. *IBM Journal of Research and Development*, 27(6):549–557, 1983.
- [41] B.E. Stewart, D. Al-Khalili, and C. Rozon. Defect Modelling and Testability Analysis of BiCMOS Circuits. *Can. J. Elec. & Comp. Eng.*, 16(4):148–153, 1991.
- [42] I.G. Tabakow. Using D-Algebra to Generate Tests for m-Logic Combinational Circuits. *International Journal of Electronics*, 75(5):897–906, 1993.
- [43] Moiez A. Tapia and Tayeb A. Guima. Multivalued Differential Calculus and Its Applications in Fault Analysis. *International Journal of Electronics*, 63(2):185–196, 1987.
- [44] G.S. Visweswaran, Akhtar uz-zaman M. Ali, Parag K. Lala, and Carlos R. P. Hartmann. The Effect of Transistor Source-to-Gate Bridging Faults in Complex CMOS. *IEEE Journal of Solid-State Circuits*, 26(6):893–896, June 1991.

- [45] Z. Vranesic. Applications and Scope of Multiple-Valued LSI Technology. In *Proceedings of Compcon (spring), (San Francisco, U.S.A.)*, pages 213–216, February 1981.
- [46] Hui Min Wang, Chung Len Lee, and Jwu E. Chen. Complete Test Set for Multiple-Valued Logic Networks. In *Proceedings of the Twenty-Fourth International Symposium on Multiple-Valued Logic*, pages 289–296, May 1994.
- [47] Michael Whitney and Jon Muzio. Decisive Differences and Partial Differences for Stuck-at Fault Detection in MVL Circuits. In *Proceedings of the Eighteenth International Symposium on Multiple-Valued Logic*, pages 321–328, May 1988.
- [48] M.E. Zaghoul and D. Gobovic. Fault Simulation in CMOS VLSI Circuits. *IEE Proceedings-E*, 138(4):203–212, 1991.

Vita

- Maher Mohammed Al-Sharif
- Born in 1969 in Khamis Mushait, Saudi Arabia.
- Received the Bachelor of Science degree in Computer Engineering in 1992 from King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia.
- Received the Master of Science degree in Computer Engineering in 1998 from King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia.