TECHNISCHE UNIVERSTIÄT MÜNCHEN

Lehrstuhl für Entwurfsautomatisierung

Test Set Optimization for Industrial SRAM Testing

Michael Linder

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor Ingenieurs

genehmigten Dissertation.

Vorsitzende: Univ.-Prof. Dr. rer. nat. Doris Schmitt-Landsiedel

Prüfer der Dissertation:

1. Univ.-Prof. Dr.-Ing. Ulf Schlichtmann
2. Associate Prof. Said Hamdioui, Ph.D.,
   Delft University of Technology / Niederlande
3. Prof. Dr.-Ing. Dr. h.c. Alfred Eder, Hochschule Augsburg

Die Dissertation wurde am 06.11.2012 bei der Technischen Universität München eingereicht und von der Fakultät für Elektrotechnik und Informationstechnik am 13.09.2013 angenommen.

Acknowledgments

This project was done during the time when I was applied as a PhD student at the department of electrical engineering at the Hochschule Augsburg in cooperation with Infineon Technologies AG and Technische Universität Müchen. The study was commissioned and completely financed by Infineon Technologies AG. Special tanks to the examiners Prof. Dr.-Ing. Alfred Eder, who supervised the project and reviewed this work, and to Prof. Dr.-Ing. Ulf Schlichtmann for his commitment and his support as reviewer.

Furthermore I want to thank everyone at Infineon who was involved into the project and who gave me all the support and information I needed to perform the study as a part of productive memory testing. Thanks to the whole team of productive memory testing (PTE) and especially to Dr.-Ing. Martin Huch for his advices and to Dipl.-Ing. (FH) Klaus Oberländer, who initialized the project and acted as supervisor at Infineon. His support enabled me to become a part of the team.

Very special thanks to Prof. Dr. ir. Ad van de Goor and Prof. Dr. ir. Said Hamdioui who I got to know during the project for their helpful advises and lead through memory testing. And finally, special thanks to my parents, Gerlinde and Bernhard Linder. Without their support, this project would not have been realized.

Abstract

Memory testing has always been an important task since semiconductor memories are commonly used. Much has been published on functional fault models and memory test algorithms. With ongoing development of memories and shrinking technology, more and more new variants of fault models arose and recent tests have been developed.

For this project the unique opportunity opened up to carry out a comprehensive analysis of memory test algorithms and faults on real productive test data. The study was initialized and enabled by Infineon Technologies AG and the tests took place on embedded SRAMs of 32-bit microcontroller devices. During full production, comprehensive tests on the embedded SRAMs could be executed, as due to improved Burn-In and test procedures extended tests have been made possible during 12 hours of Burn-In. The most important expectations on this project are new findings about detectability of memory faults and effectiveness of memory test algorithms, with the overall aim of test set optimization.

In the beginning a comprehensive study on memory faults, test algorithms and the existing test hard- and software at Infineon was done. The present potential of embedded testing was analyzed and the possible potential was identified.

To reach the project objectives 30 different march test algorithms have been combined to a study test set and were implemented into the productive test flow and performed several times at different temperature and supply voltage. The results are logged and collected in a huge data base. Via pure data mining of these productive test results, the analyses for this study could be executed. The effectiveness of memory test algorithms and the importance and effects of environmental and algorithmic stress parameters could be worked out and analyzed. A few algorithms could be identified that are outstanding and most effective to cover the largest part of

different fault models. In addition to the analysis of single test algorithms, the relationship of combinations of test algorithms was analyzed. An efficient combination of test algorithms avoids redundant testing and keeps the test time as short as possible. Through evaluating the results of test algorithms pairwise, efficient and inefficient combinations could be identified and a new approach to rate the quality of effectiveness of test algorithms could be established. These findings are important for the selection of algorithms for productive memory test sets - moreover the same results could be used to classify test algorithms and functional fault models, where algorithms with similar properties are allocated to a specific subset of functional fault models.

With help of heuristic logic minimization, a formal approach to test set optimization was established. The test results could be processed and the set of essential test algorithms could be determined that fulfills two requirements at the same time: maximum fault coverage and minimum test effort. By weighted ordering of these essential algorithms, a "function" could be generated that relates fault coverage to test length. This is important for test set development, as desired yield and expected test time can be estimated. Depending on test requirements and desired yield, a subset of essential algorithms can be chosen and the necessary test time can be estimated. The analysis has shown that already the combination of three test algorithms is able to detect nearly 98% of faults at a minimum of test time.

Due to the fact that the comprehensive test set was performed during Burn-In, which causes artificial aging, the influence of high temperature and high voltage stress could also be analyzed and interesting results could be observed. Before and after the stress phase during Burn-In, tests have been executed at similar environmental test conditions. Hence, the results are comparable and the influence of Burn-In stress on fault manifestation could be worked out. The analysis shows that Burn-In stress increases the number of

faults about four times. Especially for highly safety critical products, Burn-In has a not negligible effect in memory testing. Without Burn-In these faults remained undetected and latent, and would possibly appear during life time. Not only the number of faults increases due to stress, also the behavior and manifestation of some faults changes. By classifying the faults before and after Burn-In, a shift from dynamic to static faulty behavior became observable. The same faults that appeared as dynamic before stress, manifested as static afterwards. This finding influences the selection of test algorithms for different test sets before and after Burn-In or at wafer test.

The results of this project are directly used to improve the memory test process at Infineon and some findings could already be fed back into the development of Built-In Self-Tests and the productive test flow.

Kurzfassung in deutscher Sprache

Seit der massenhaften Verbreitung von Halbleiter Speichern ist auch deren Test immer ein wichtiges Thema gewesen. Seitdem wurde ständig daran gearbeitet und viel zu funktionalen Fehlermodellen und Testalgorithmen veröffentlicht. Mit fortschreitender Entwicklung und immer kleiner werdenden Strukturen wurden auch immer mehr neue Fehlermodelle entdeckt und neuere Tests entwickelt.

Für das Projekt, das in dieser Arbeit beschrieben wird, hat sich die einmalige Gelegenheit ergeben eine umfangreiche Untersuchung von Fehlern und Testalgorithmen, basierend auf produktiven Messdaten vorzunehmen. Die gesamte Studie wurde durch die Infineon Technologies AG ermöglicht und auch die Untersuchungen der eingebetteten SRAMs in 32-bit Mikrocontrollern fanden dort statt. Durch ein kombiniertes Test und Burn-In System wurden umfangreiche Tests der Speicher vor und nach einer 12-stündigen Burn-In Phase ermöglicht. Ziel dieser Studie war neue Erkenntnisse über die Detektierbarkeit von Speicherfehlern zu gewinnen und, vor dem Hintergrund die Zusammenstellung von Testsets zu optimieren, die Effektivität von Testalgorithmen zu untersuchen.

Zu Beginn der Studie wurde eine ausführliche Recherche zu Fehlern in Halbleiterspeichern, Testalgorithmen und der Testhard- und software bei Infineon durchgeführt. Dazu wurde die aktuell Konfiguration und die potentiell mögliche Konfiguration des integrierten Testsystems analysiert.

Um diese Ziele zu erreichen wurde ein Set von 30 verschiedenen Testalgorithmen erstellt, das anschließend in den produktiven Testablauf eingeflochten wurde und bei verschiedenen Spannungen und Temperaturen wiederholt ausgeführt wurde. Die Ergebnisse dieser Tests wurden für die weitere Analyse in einer Datenbank gespeichert, auf deren Basis die Datenanalyse für diese Studie stattfand. Somit war

es möglich die Effizienz von Testalgorithmen und den Effekt, den verschiedene algorithmische Stressparameter oder Umgebungsparameter haben, herauszuarbeiten und zu analysieren. Einige Algorithmen konnten identifiziert werden, die in ihrem Verhalten herausstechen und mit deren Hilfe die meisten Fehlermodelle bereits gefunden werden können. Zusätzlich zur Analyse der Effizienz einzelner Testalgorithmen, wurde auch deren Kombination untersucht. Denn eine Effiziente Kombination aus Algorithmen in einem Testset vermeidet redundante Tests und die Testzeit kann so verringert werden. Durch den Vergleich der Testergebnisse von jeweils zwei Algorithmen konnten effiziente und ineffiziente Paarungen identifiziert werden und ein Ansatz zur Bewertung der Effizienz von Testalgorithmen wurde entwickelt. Die so gewonnenen Erkenntnisse sind wichtig für die Auswahl von Testalgorithmen für produktive Testsets. Überdies wurden die Ergebnisse aus dieser Analyse auch dafür verwendet, Algorithmen und Fehlermodelle zu klassifizieren, wobei Algorithmen mit ähnlichen Eigenschaften zusammengefasst und Gruppen von bestimmten Fehlermodellen zugeordnet wurden.

Mit Hilfe heuristischer Logikminimierung wurde ein Ansatz für die formale Testset Optimierung entwickelt. Die Testergebnisse der Messungen wurden so verarbeitet, dass ein Set von essentielle Algorithmen ermittelt wurde, das sowohl die Anforderung an maximale Fehlerausbeute als auch an minimale Anzahl an nötigen Algorithmen erfüllt. Durch Gewichten dieser essentiellen Algorithmen wurde eine Zuordnung von Fehlerausbeute zu Testlänge in Form einer „Funktion“ möglich. Dies ist vor Allem für die Testset Entwicklung interessant, da abhängig von den Testanforderungen, die Algorithmen für eine gewünschte Ausbeute selektiert werden können und die notwenige Testzeit dafür dann abgeschätzt werden kann. Die Ergebnisse dieser Studie zeigen, dass bereits mit einer Kombination aus

nur drei Testalgorithmen nahezu 98% der Fehler entdeckt werden, wobei die Testzeit dabei relativ gering ist.

Durch die Tatsache, dass die Untersuchungen in dieser Studie während des Burn-In Tests stattfanden, konnten auch die Auswirkungen von Hochtemperatur- und Hochspannungsstress auf das Fehlerverhalten der Speicher untersucht werden – mit interessanten Ergebnissen. Vor und nach der Stressphase wurden die Tests bei ähnlichen Umgebungsparametern durchgeführt. Somit war es möglich diese Ergebnisse miteinander zu vergleichen und so den Effekt, den Burn-In auf das Fehlerverhalten hat herauszuarbeiten. Die Analyse zeigt, dass sich die Anzahl der gefundenen Fehler nach Burn-In etwa vervierfacht hat. Besonders für sicherheitstechnisch relevante Produkte heißt das, dass Burn-In einen nicht zu vernachlässigenden Einfluss auf das Testen von Halbleiterspeichern hat. Ohne Burn-In wären diese Fehler unentdeckt geblieben und hätten sich erst im Laufe der Zeit manifestiert und so zu Ausfällen geführt. Aber nicht nur die Anzahl der Fehler stieg an sondern auch das Verhalten einzelner Fehler hat sich durch Burn-In verändert. Es wurde eine Veränderung von dynamischen Fehlern vor Burn-In hin zu statischem Verhalten beobachtet. Dieselben Fehler, die vorher als dynamisch klassifiziert wurden, haben sich durch Burn-In als statische Fehler manifestiert. Diese Erkenntnis beeinflusst die Selektion von Testalgorithmen für unterschiedliche Testsets vor und nach Burn-In oder bereits für das Wafertesten.

Ergebnisse dieser Studie konnten bereits unmittelbar genutzt werden um Speichertests bei Infineon zu verbessern und einige Erkenntnisse konnten auch in die Entwicklung des integrierten Selbsttests für Produktionstests zurückgeführt werden.

Contents

Chapter 1 Introduction ... 1
1.1 Previous Work ... 2
1.2 Motivation ... 4
1.3 Planned Work and Project Outline ... 4
1.4 Semiconductor Memories ... 6
1.4.1 Memory Technology ... 7
1.4.2 Static Random Access Memories ... 9

Chapter 2 Memory Faults ... 13
2.1 Definitions ... 14
2.2 Classification of Memory Faults ... 14
2.2.1 Static versus Dynamic Faults ... 15
2.2.2 Simple versus Linked Faults ... 16
2.2.3 Single-cell versus Coupling Faults ... 17
2.2.4 Address Decoder Faults ... 17
2.2.5 Peripheral Faults ... 18
2.3 Fault Primitives and Functional Fault Models ... 19

Chapter 3 Memory Test Algorithms ... 25
3.1 Nomenclature ... 26
3.2 Test Algorithms ... 27
3.3 Algorithmic Test Parameters ... 31
3.3.1 Address Direction ... 31
3.3.2 Addressing Mode ... 32
3.3.3 Data Background ... 34
3.4 Environmental Parameters ... 35

Chapter 4 Test Environment and Setup ... 37
4.1 Memory Testing ... 38
4.1.1 MSIST ... 38
4.1.2 MBIST ... 39
4.2 MBISTPLUS ... 40
4.2.1 Potential of MBISTPLUS V3.0 ... 43
4.2.2 Potential of MBISTPLUS V4.2 ... 44
4.3 Burn-In ... 45

4.4 Test Strategy 46
4.5 Test Setup 47
4.5.1 Tested Devices and Memories 48
4.5.2 Study Test Set 49
4.5.3 Test Environment 51
4.5.4 Test Flow 52
4.5.5 Data Acquisition 54

Chapter 5 Fault Coverage of Test Algorithms 57
5.1 Definitions 58
5.1.1 Fault Coverage 58
5.1.2 Test 58
5.1.3 Test Set 58
5.2 Fault Coverage at Different Environmental Conditions 59
5.2.1 Test Results 59
5.2.2 Data Evaluation 61
5.3 Fault Coverage of Test Algorithms 63
5.3.1 Test Results 63
5.3.2 Evaluation of Fault Coverage 66
5.3.3 Unique Faults 67
5.4 Influence of Algorithmic Test Parameters 69
5.5 Summary and Conclusions 73

Chapter 6 Effectiveness of Test Algorithms 75
6.1 Definitions 76
6.1.1 Union and Intersection 76
6.1.2 Subsets and Coverage 80
6.2 Evaluation Method 81
6.3 Efficient Pairs of Algorithms 82
6.3.1 Test Results 82
6.3.2 Data Evaluation 84
6.4 Classification of Algorithms 86
6.4.1 Similar Fault Coverage and Subsets 87
6.4.2 Characteristic March Elements 94
6.4.3 Grouping and Classification 100
6.4.4 Consistencies and Inconsistencies 102

6.5 Estimation of Fault Distribution 103
6.6 Summary and Conclusions 106

Chapter 7 Test Set Optimization 109
7.1 Formal Optimization 110
7.2 Test Data Preparation 110
7.3 Test Results 112
7.3.1 Essential Algorithms 112
7.3.2 Fault Coverage related to Test Length....... 115
7.4 Summary and Conclusions 119

Chapter 8 Variation of Fault Manifestation 121
8.1 Setup and Environment 122
8.2 Increase of Fault Coverage 123
8.3 Variation of Fault Behavior 125
8.3.1 Data Evaluation Technique 126
8.3.2 Test Results 127
8.4 Summary and Conclusions 130

Chapter 9 Perspectives.. 131

References 135

Additional Literature 139

Appendix A Additional Results of Chapter 6 143
A.1 Fault Coverage of Algorithms 143
A.2 Effectiveness of Pairs of Algorithms 153

Appendix B Calculation of Fault Distribution 165

Appendix C ESPRESSO Algorithm and Software 167
C.1 ESPRESSO Heuristic Algorithm 167
C.2 ESPRESSO Software 168

Appendix D Additional Results of Chapter 7 171

List of Figures

Figure 1.1. Block diagram of a memory.................................... 7
Figure 1.2. Types of semiconductor memories......................... 8
Figure 1.3. Properties of different types of memories 9
Figure 1.4. 6-Transistor SRAM cell... 10
Figure 1.5. SRAM cell array.. 11
Figure 2.1. Classification of functional fault models 15
Figure 2.2. Linked memory faults 16
Figure 2.3. Address decoder faults.. 18
Figure 3.1. Addressing of one- and more-dimensional test algorithms 30
Figure 3.2. Addressing directions... 32
Figure 3.3. Address modes 33
Figure 3.4. Data background patterns 34
Figure 3.5. Buggy background patterns 35
Figure 4.1. DUT block diagram using MSIST 39
Figure 4.2. DUT block diagram using MBIST......................... 40
Figure 4.3. Bathtub curve.. 45
Figure 4.4. TC1797 block diagram 48
Figure 4.5. Memory test flow 51
Figure 4.6. Test flow 52
Figure 4.7 Test numbers and environmental conditions 53
Figure 4.8. IBIS flow and data acquisition.............................. 55
Figure 5.1. Fault coverage... 60
Figure 5.2. Fault coverage and exclusive faults...................... 61
Figure 5.3. Fault coverage of algorithms 65
Figure 5.4. Fault coverage of algorithmic parameters........... 71
Figure 6.1. Union and intersection 77
Figure 6.2. Efficiency of pairs of algorithms 78
Figure 6.3. Color key 79
Figure 6.4. Special cases of union and intersection................ 79
Figure 6.5. Subsets of fault coverage....................................... 81
Figure 6.6. Efficient and inefficient algorithms 84
Figure 6.7. Venn diagrams I 88

Figure 6.8. Venn diagrams II.....89
Figure 6.9. Venn diagrams III.....91
Figure 6.10. Venn diagrams IV.....92
Figure 6.11. Venn diagrams V.....93
Figure 6.12. Venn diagrams VI.....94
Figure 6.13. Fault distribution.....105
Figure 7.1. System representing algorithms and faults.....110
Figure 7.2. ESPRESSO output file of TN6531.....113
Figure 7.3. Fault coverage over test length.....117
Figure 8.1. Test flow surrounding Burn-In.....122
Figure 8.2. Fault distribution at low voltage.....124
Figure 8.3. Fault distribution at high voltage.....125
Figure 8.4. Fault model distribution a low voltage.....128
Figure 8.5. Fault model distribution at high voltage.....129
Figure A.1. Fault coverage at TN1522.....144
Figure A.2. Fault coverage at TN1622.....145
Figure A.3. Fault coverage at TN6531.....146
Figure A.4. Fault coverage at TN6631.....147
Figure A.5. Fault coverage at TN3741.....148
Figure A.6. Fault coverage at TN3841.....149
Figure A.7. Fault coverage at TN3941.....150
Figure A.8. Fault coverage at TN4441.....151
Figure A.9. Fault coverage at TN4541.....152
Figure C.1. ESPRESSO input and output file.....168
Figure D.1. Fault coverage over test length at TN1522.....172
Figure D.2. Fault coverage over test length at TN1622.....173
Figure D.3. Fault coverage over test length at TN6531.....174
Figure D.4. Fault coverage over test length at TN6631.....175
Figure D.5. Fault coverage over test length at TN3741.....176
Figure D.6. Fault coverage over test length at TN3841.....177
Figure D.7. Fault coverage over test length at TN3941.....178
Figure D.8. Fault coverage over test length at TN4441.....179
Figure D.9. Fault coverage over test length at TN4541.....180

List of Tables

Table 2.1. Single-cell FPs and FFMs 19
Table 2.2. Two-cell FPs and FFMs 21
Table 3.1 Symbols and Nomenclature 27
Table 3.2. Memory test algorithms 29
Table 4.1. Properties of MBISTPLUS 41
Table 4.2. Memory test algorithms 42
Table 4.3. Study test set 49
Table 4.4. Test numbers and environmental conditions 53
Table 5.1. Fault coverage per test number 60
Table 5.2. Fault coverage of algorithms 64
Table 5.3. Fault coverage of algorithmic test parameters 70
Table 6.1. Effectiveness of pairs of algorithms 83
Table 6.2. Fault coverage and Q_{Eff} 88
Table 6.3. March algorithms for simple coupling faults 96
Table 6.4. March algorithms for linked faults 98
Table 6.5. March algorithms for dynamic faults 99
Table 6.6. Fault model coverage of traditional march tests 100
Table 6.7. Sets of algorithms and functional faults 101
Table 6.8. Determination of fault distribution 105
Table 6.9. Fault distribution 105
Table 7.1. Algorithm-Fault truth table 111
Table 7.2. Essential algorithms for TN6531 114
Table 7.3. Fault coverage over test length 116
Table 8.1. Fault coverage during Burn-In 123
Table 8.2. Algorithm-fault-allocation 126
Table 8.3. Fault model distribution at low voltage 127
Table 8.4. Fault model distribution at high voltage 128
Table A.1. Fault coverage at TN1522 144
Table A.2. Fault coverage at TN1622 145
Table A.3. Fault coverage at TN6531 146
Table A.4. Fault coverage at TN6631 147
Table A.5. Fault coverage at TN3741 148
Table A.6. Fault coverage at TN3841 149

Table A.7. Fault coverage at TN3941 150
Table A.8. Fault coverage at TN4441 151
Table A.9. Fault coverage at TN4541 152
Table A.10. Values of Q_{Eff} at TN6531 154
Table A.11. Union and intersection at TN1522 155
Table A.12. Union and intersection at TN1622 156
Table A.13. Union and intersection at TN6531 157
Table A.14. Union and intersection at TN6631 158
Table A.15. Union and intersection at TN3741 159
Table A.16. Union and intersection at TN3841 160
Table A.17. Union and intersection at TN3941 161
Table A.18. Union and intersection at TN4441 162
Table A.19. Union and intersection at TN4541 163
Table D.1. Fault coverage over test length at TN1522..........172
Table D.2. Fault coverage over test length at TN1622..........173
Table D.3. Fault coverage over test length at TN6531..........174
Table D.4. Fault coverage over test length at TN6631..........175
Table D.5. Fault coverage over test length at TN3741..........176
Table D.6. Fault coverage over test length at TN3841..........177
Table D.7. Fault coverage over test length at TN3941..........178
Table D.8. Fault coverage over test length at TN4441..........179
Table D.9. Fault coverage over test length at TN4541..........180

Abbreviations and Symbols

2^i	see: POI
AF	Address decoder fault
BI	Burn-In
BL	Bit-line
$\overline{BL}$	Inverted bit-line
C11	130 nm technology
cb	Checkerboard data background
CF	Coupling fault
CFdrd	Deceptive read destructive coupling fault
CFds	Disturb coupling fault
CFid	Idempotent coupling fault
CFin	Inversion coupling fault
CFir	Incorrect read coupling fault
CFrd	Read destructive coupling fault
CFrr	Random read coupling fault
CFrrd	Random read destructive coupling fault
CFst	State coupling fault
CFtr	Transition coupling fault
CFud	Undefined disturb coupling fault
CFur	Undefined read coupling fault
CFus	Undefined state coupling fault
CFuw	Undefined write coupling fault
CFwd	Write destructive coupling fault
CPU	Central Processing Unit
cs	Column-stripe data background
D	Delay between march elements
DDR	Double Data Rate SDRAM
DMI	Data Memory Interface
DRDF	Deceptive read destructive fault
DRF	Data retention fault
DUT	Device Under Test
ECC	Error Correcting Code
EEPROM	Electrically Erasable PROM

EPROM	Erasable PROM
F	Set of faults
$\|F\|$	Cardinality of a set of faults (number of faults)
FC	Fault Coverage
fx	Fast-x addressing mode
fy	Fast-y addressing mode
FFM	Functional Fault Model
FSM	Finite State Machine
FP	Fault Primitive
DRAM	Dynamic Random Access Memory
I	Intersection
IBIS	Interconnect Built-In Self-Test
IFA	Inducted Fault Analysis
IRF	Incorrect read fault
JTAG	Joint Test Action Group
L90	90 nm technology
LF	Linked Fault
n	Number of operations (length of a test algorithm)
MBIST	Memory Built-In Self-Test
MBIST+	see: MBISTPLUS
MBISTPLUS	Infineon MBIST
ME	March Element
MUT	Memory Under Test
MSIST	Memory Software-Implemented Self-Test
NAF	No access fault
#O	Number of Operations
PMI	Program Memory Interface
POI	Power of i (addressing mode)
PROM	Programmable ROM
Q_{Eff}	Quotient of Efficiency
r	Read operation
RAR_{wa}	walking read-after-read sequence
RDF	Read destructive fault
ROM	Read-Only Memory

RRDF	Random read destructive fault
RRF	Random read fault
rs	Row-stripe data background
SAF	Stuck-at fault
SDRAM	Synchronous DRAM
SF	State fault
so	Solid data background
SoC	System-on-a-Chip
SOS	Sensitizing Operation Sequence
SQL	Structured Query Language
SRAM	Static Random Access Memory
SS	Simple static single-cell fault
TF	Transition fault
TL	Test Length
TN	Test Number
TRF	Test Result File
TT	Test Time
U	Union
URF	Undefined read fault
USF	Undefined state fault
UWF	Undefined write fault
w	Write operation
WDF	Write destructive fault
WL	Word-line
0	Logic zero
1	Logic one
( )	Delimiter of march elements
{ }	Delimiter of march algorithms
⇳	Arbitrary address direction
⇧	Up address direction (lowest to highest)
⇩	Down address direction (highest to lowest)
$\subset$	… is subset of …
$\tilde{\subset}$	… is approximately subset of …

Chapter 1
Introduction

In modern microcontroller devices for highly safety critical applications, it is essential to ensure the freedom from faults for the embedded memories. Therefore a very high effort is necessary to reach the aim of "zero defects". For this project, the unique opportunity to perform a comprehensive analysis of memory tests became possible at Infineon Technologies AG. In contrast to simulation based test optimization, in this work, a large amount of productive memory test results is used for statistical analysis and evaluation. The aims of the project are to investigate the embedded self-tests of static semiconductor memories and to analyze productive memory test results of automotive microcontroller devices with the aim to improve and optimize the selection of tests for embedded SRAMs.

The project became possible because during productive testing, extended test time due to Burn-In was available for comprehensive tests and analysis. Hence the project and the whole test setup and analysis were closely associated with the productive memory test flow.

1.1 Previous Work

Much work has been published on memory testing (e.g. [1]-[31]). Before 1980, memory testing means "ad hoc testing". Long and partly complex test patterns have been applied with the hope to detect as many faults as possible as fault models and proofs did not exist. Typical tests of this period are GALPAT or Walking 1/0, where the test time is extremely long and the test time of those tests grows quadratically with the memory size.

During the early 1980s, functional fault models (FFMs) have been introduced. The tests that have been developed on basis of FFMs are typically of order O(n), i.e. linear with the memory size, where n is the memory size, and the desired fault coverage of these algorithms could mathematically be proven. Important FFMs have been stuck-at faults (SAFs), address decoder faults (AFs), coupling faults (CFs) and neighbourhood pattern sensitivity faults (NPSFs). Functional fault models are abstract and reflect the faulty behaviour of a memory independently of technology or real design. At that time also the inductive fault analysis (IFA) was used to establish new fault models based on simulated designs. More and more fault models have been developed and so also new march tests. For the first time, Ad van de Goor sums up the previous work on memory testing in his book [1] in 1998. It is a still important work that comprehensively treats memory testing, memory test algorithms and functional fault models.

However, not all faults could be explained with the existing set of functional fault models. Hence, during the late 1990s, the concept of fault primitives was introduced. Due to the complete description of fault primitives, the memory faults could be classified and additional functional fault models could be described such as write disturb faults, incorrect read faults or transition CFs. An important work, especially concerning the definition of memory faults, fault

models and fault primitives was done by Said Hamdioui, who sums up his experience in his book [2] in 2004.

In [3] and [4], conditions for march sequences are described that have to be fulfilled to cover specific types of functional faults. Most publications on memory testing use these kinds of mathematical proofs of fault coverage to describe the performance and effectiveness of memory test algorithms. Comparatively rarely, real test results are presented in published work. Hamdioui et al. and Al-Ars et al. are presenting some industrial results in [3, 5, 6] and evaluate the test performance of different algorithmic tests and stress combinations. Additionally an approach to test set optimization is depicted, where the entire set of tests is reduced to a minimum necessary set of algorithms and stress combinations. The results of all algorithms are compared to each other in such a form that union and intersection of detected faults are determined and listed. This approach is also taken for a major part of the fault analysis in this work. However, the number of results presented in previous work is insufficient for a comprehensive test set optimization, and also the set of test algorithms that are taken into account is small.

The performance of memory test algorithms is usually proven mathematically and the expected fault coverage is determined (e.g. [7, 8, 9]). However, in practice, there is only a simple fail or pass information and the fault model which is behind this fail is unknown. So, the selection of test algorithms can hardly be based on theory only, but experimental results have to be used for an efficient selection of test algorithms.

1.2 Motivation

Many analyses and investigations have been done on memory test algorithms and functional fault models as described above. However, for industrial purpose the usability of test algorithms described in literature needs to be verified and efficient combinations of tests have to be identified for optimized test performance at industrial and productive semiconductor memory testing.

Therefore, a comprehensive analysis on the efficiency of test algorithms and test parameters is necessary to determine new and efficient test sets to fulfill two main requirements for industrial semiconductor memory testing:

- high fault coverage, and
- low test effort.

This analysis was done as a project in cooperation with Infineon Technologies, who provided a productive test environment for such comprehensive tests and statistical analyses. The planed outcome of the project is an optimization for productive testing of embedded semiconductor memories. The planed contents of this work are described in the following section.

1.3 Planned Work and Project Outline

The intended content of this work was discussed with Infineon throughout the project, as both the course and the results of the investigation were not completely foreseeable in the beginning. For the initial definition of the project, the following items have been panned:

- Understand SRAM test concepts (SIST, MBIST+ various versions and generations) and test algorithms.
- Carry out a literature survey on known SRAM technological problems, fault and failure causes.
- Analysis of IBIS flow (Burn In) and current test flow.
- Get basic knowledge about system of automotive microcontrollers, TC1797/TC1767, TC1787 … at least for MBIST+ / test access and access to memories through JTAG and system (mapping).
- Comprehensive investigations about test algorithms, address algorithms, backgrounds, SRAM scrambling involvement, system scrambling involvement.
- Analysis of environmental conditions (temperature, supply voltage) and other parameters (programmable self timings, weak write driver, digital margin mode) of SRAM and ROM.
- Setup exhaustive test plan for TC1797/TC1767 and TC1787.
- Setup database and visualization in html for tracking test results of exhaustive tests.
- Create test setup for application in test flow using MBIST+ and drive implementation of these tests in test and IBIS flow.
- Accumulate results in a database.
- Map possible technological causes with observation findings.
- Evaluate results for most efficient algorithms and test conditions.
- Apply learning from 130nm technology (C11) to 90nm technology (L90) as there are more degrees of freedom in test generation in L90.
- Influence future direction of MBIST+ design with learning outcome.
- Minimize test time vs. effectiveness for C11 and L90 test flow using all parameters available.

Not all of these items could be completed during the project as for some reasons concerning the productive test flow, the test setup for L90 could not be achieved completely and the number of devices that could be tested for L90 was not enough for meaningful and comparable statistics. Hence the investigations of the project are focused on the results of C11 testing.

However, during the project new aspects came into account and new ideas for analyses have been developed. So, the relation of faults and test algorithms has been analyzed and the distribution of different fault models within the tested memories was estimated based on the obtained test results. Also, variations of fault manifestation during Burn-In became visible and have been analyzed in more detail.

The major outcomes of the project are:

- Comprehensive literature survey on memory testing, test algorithms and memory faults.
- Development of a suitable test strategy and setup and implementation of test to obtain productive memory test results.
- Analysis of effectiveness of memory test algorithms.
- Estimation of fault distribution after wafer test.
- Test set optimization.
- Analysis of variation of fault manifestation during Burn-In.

1.4 Semiconductor Memories

Semiconductor memories are very important storage elements in electronic devices. This project is entirely related to embedded SRAMs of microcontroller devices. As a short introduction to memory technology, this chapter gives an overview about the basic technological background of

semiconductor memories and Static Random Access Memories (SRAMs) in particular.

1.4.1 Memory Technology

Memories are part of most electronic devices to store program information or data. Usually the information is represented by ones and zeros, i.e. high and low potential in the electrical model. The simplest form of a block diagram for a memory is shown in Fig. 1.1 [1].

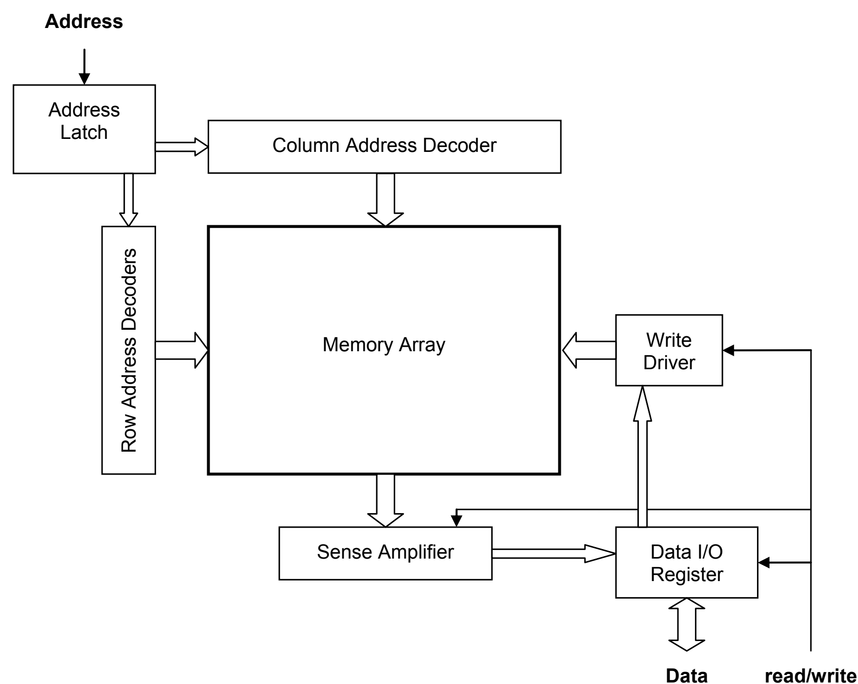

Figure 1.1. Block diagram of a memory

The basic components of a memory are the memory cell array itself which contains the actual information and peripheral logic address decoder, write drivers and sense amplifiers. The address decoder is necessary to access the

single memory cells and write driver and sense amplifier enable to write to and read data from the memory.

The types of semiconductor memories can be distinguished by their storage method, the form factor and the technology [10]. Basically volatile and non-volatile storage methods are common. Volatile means the stored information needs to be refreshed or supply voltage needs to be kept in order to keep the data, while non-volatile storage elements also keep the information without permanent supply voltage. A non-complete diagram of different and most common memory types is shown in Fig. 1.2.

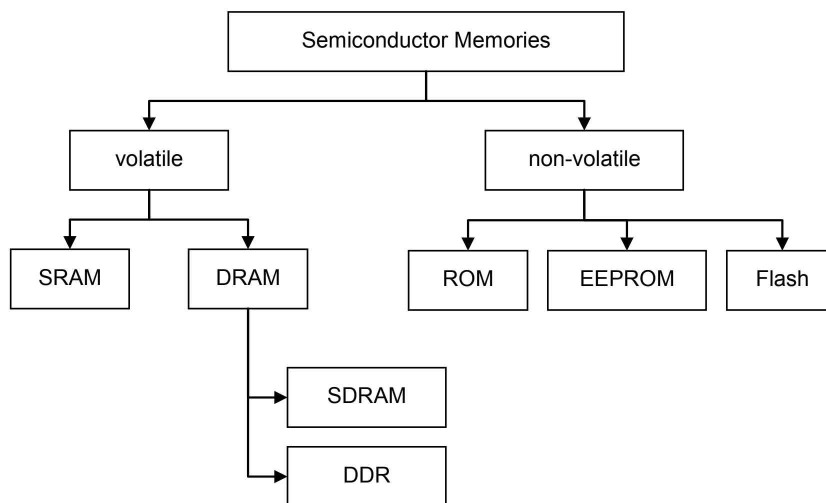

Figure 1.2. Types of semiconductor memories

Depending on the desired application for a memory, factors like density, power consumption and performance have to be taken into consideration. In [10], these factors are compared and shown in a triangle. Following to [10], Fig. 1.3 shows a few types of semiconductor memories where they are placed concerning their main properties: performance, power and density. ROM has the lowest power consumption of the

compared types. DRAM only needs low space as the density is very high but the power consumption is very high at the same time. SRAM has a good balanced ratio of power consumption, density and performance.

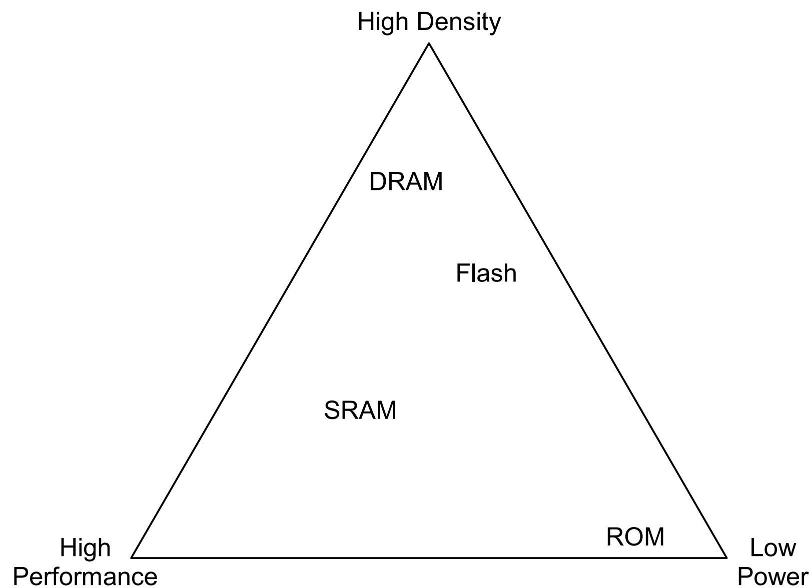

Figure 1.3. Properties of different types of memories

1.4.2 Static Random Access Memories

Static random access memories (SRAMs) are the “workhorses of memories” [10]. Besides DRAMs, SRAMs are often used in systems on a chip (SoCs) if high speed memories are needed, such as caches in microprocessors [10].

SRAMs are, as the name suggests, static. In contrast to DRAMs, static RAMs need no refreshment of the data, but retain their state until it is overwritten as long as power is supplied (volatile memory). SRAM cells can enter two different stable states which represent logical values ‘0’ and

'1'. Once the cell is in one state, it remains stable in it [2]. Basically a SRAM cell consists of two inverters that are fed back and thus stabilize themselves. The electrical model of a 6-transistor SRAM cell is shown in Fig. 1.4. It consists of four transistors forming the two inverters and two gate transistors that enable the cell to access the bit-lines (BL and inverted bit-line $\overline{BL}$) if the word-line (WL) is activated.

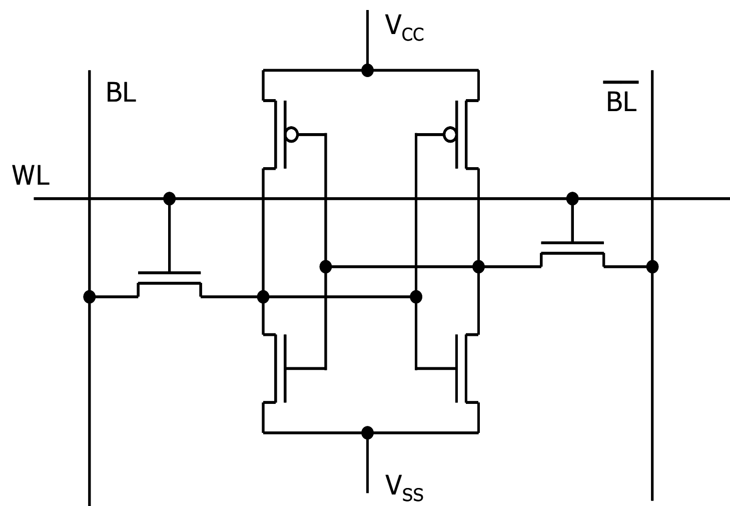

Figure 1.4. 6-Transistor SRAM cell

In a memory the cells are arranged in a regular array, where the single cells are accessed via word-line (horizontal address) and bit-lines (vertical address). Fig. 1.1 shows the memory array connected to address decoder, write driver and sense amplifier. A schematic of a regular SRAM array is shown in Fig. 1.5. A single cell can be accessed by accessing a specific word-line and bit-line. When addressing a word-line, all cells in the row are accessed. The desired cell is then selected by addressing the corresponding bit-line. Hence, one cell is uniquely identified by its horizontal and vertical address.

Note that logical and physical cell addresses usually are not the same. Due to mirroring and scrambling, the physical

and logical layout of a cell array may differ extremely from each other. In Fig. 1.5 the SRAM cells are mirrored about the x- and y-axis. A mirroring about the x-axis facilitates the use of common power supply for two adjacent rows. Mirroring about the y-axis enables an optimized bit-line layout [10]. Furthermore, mirroring facilitates sharing isolation and power supply for adjacent rows or columns.

On the one hand mirroring and scrambling is necessary because of layout considerations, and on the other hand it also may reduce interferences between adjacent cells due to compensating capacitive influence of bit- and word-lines.

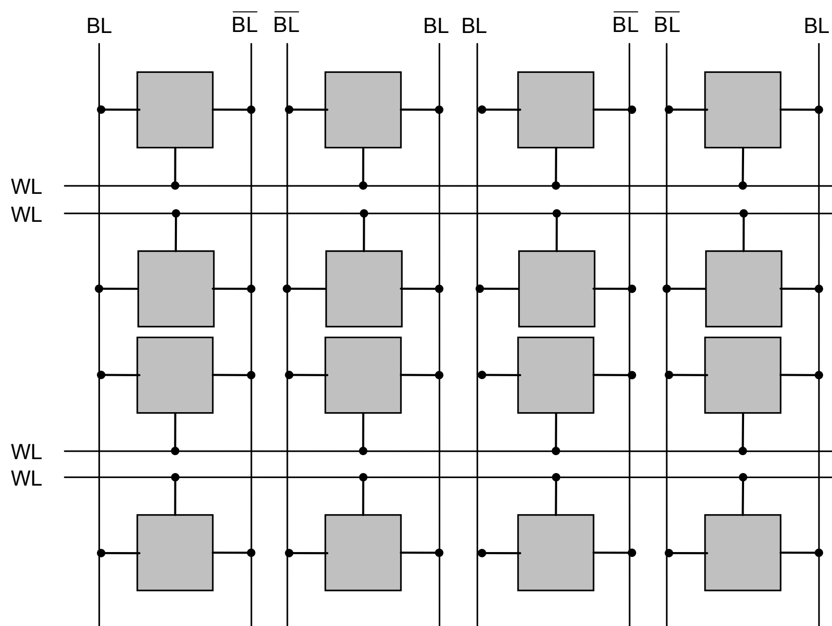

Figure 1.5. SRAM cell array

Chapter 2
Memory Faults

Memory faults often cause failing electronic devices. Wrong information stored or read from memories cause faulty behavior of the whole system. Actually physical defects cause the faults in a memory, but the defects are normally not visible to the outside. Only the faulty behavior is recognizable as functional fault.

In this chapter a short introduction to memory faults and the definition of fault primitives and functional fault models is given. Fault primitives and functional fault models are base for a targeted memory test development.

2.1 Definitions

The concepts of fault primitives and functional fault models are described in [2] and [11]. Accordingly, the following definitions apply.

Fault primitives (FPs) describe the sensitizing operation sequence (SOS) and the corresponding faulty behavior of a certain fault. The SOS is a sequence of operations applied to the memory that results in faulty behavior. A fault primitive is denoted as <S/F/R>. S describes the SOS that sensitizes the fault; F describes the fault, i.e. the value or faulty behavior of the memory cell (e.g. the cell flips). R describes the logic output value of a read operation [2].

The concept of FPs allows to create the set of functional fault models (FFMs), which are defined as a non-empty set of fault primitives that inherit the properties of the FPs [2]. Both, FPs and FFMs, describe the faulty behavior of a memory cell and do not describe physical defects. Defects are the physical reason for a fault and, depending on technology, memory type, and other hardware reasons, there may be different types of defects that cause specific FPs.

2.2 Classification of Memory Faults

Based on the SOS and faulty behavior, the FPs and FFMs can be classified according to [2]:

1. the number of sequential operation in the SOS, into static and dynamic faults.
2. the way of manifestation, into simple and linked faults.
3. the number of different cells involved, into single-cell and multi-cell (coupling faults and neighborhood pattern sensitivity faults).

These classifications are independent of each other, as the factors of the SOS are independent. The classification of FFMs [2, 1] is summarized in Fig. 2.1.

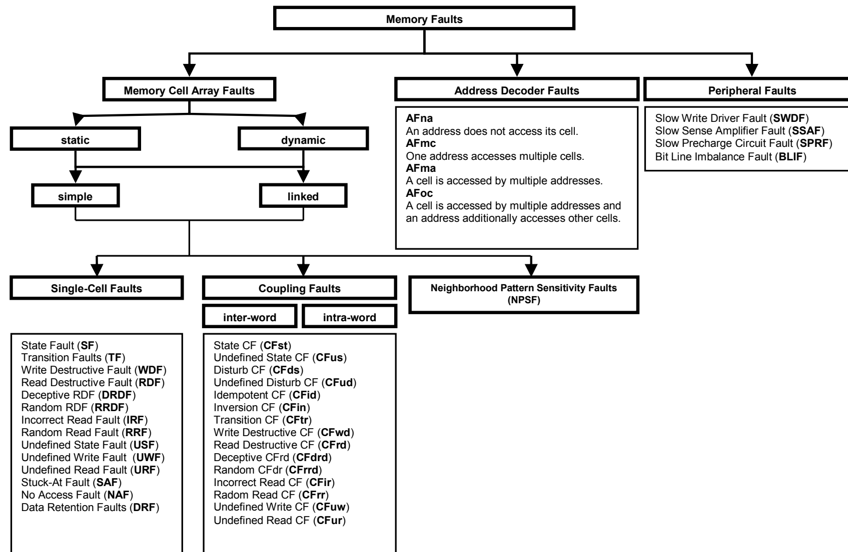

Figure 2.1. Classification of functional fault models

Additional to memory cell array faults, Fig. 2.1 also shows address decoder faults and peripheral faults.

2.2.1 Static versus Dynamic Faults

Fault primitives can be divided into static and dynamic fault depending on the number of sequentially performed operation (#O) in a SOS [2].

Static FPs are always sensitized by the state of the cell (i.e. no operation) or at most one operation, i.e. #O ≤ 1. Dynamic faults are sensitized by more than one operation in the SOS, i.e. #O > 1.

2.2.2 Simple versus Linked Faults

Fault primitives can manifest themselves as simple or linked faults. A simple fault cannot influence the behavior of another one, whereas linked faults (LFs) can influence the behavior of other faults and masking can occur [1, 2, 3]. The types of linked faults are illustrated in Fig. 2.2 [3, 12]

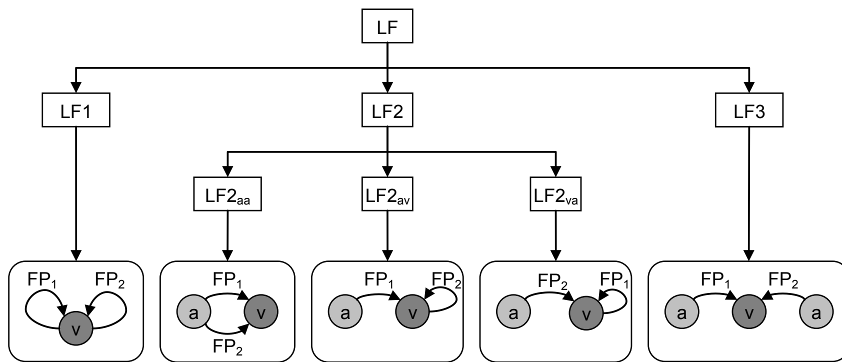

Figure 2.2. Linked memory faults

Depending on the number of faults involved, linked faults are divided into one-cell (LF1), two-cell (LF2) or three-cell (LF3) linked faults. The cells involved into a LF are called aggressor cell (a-cell) or victim-cell (v-cell), where an SOS on the a-cell causes a fault in the v-cell. I.e. the state or an operation on the aggressor causes a faulty behavior in the v-cell. Except for LF1, at least two a-cells are involved into a LF, but always only one cell is the v-cell. With LF2, the v-cell is the second a-cell at the same time and influences itself, where with LF3, there are two different a-cells beside the v-cell.

For all types of linked faults, there are two FPs that influence the behavior of the v-cell in such a way, that the second FP may mask the fault of the first FP. Masking means the second fault on the same cell reverses the effect of the first fault. For example: an operation on an aggressor a-cell causes the v-cell to flip and a second operation on another a-cell

causes the v-cell to flip back (LF3), then the fault on the v-cell is masked. LF2s can furthermore be divided depending on which cell (a- or v-cell) the first and second FP appears into $LF2_{aa}$, $LF2_{av}$ and $LF2_{va}$. At a $LF2_{aa}$ linked fault, both FPs are coming from the a-cell, where at a $LF2_{av}$ linked faults, the first FP comes from the a-cell and the second FP comes from the v-cell, and vice versa at a $LF2_{va}$ linked fault.

2.2.3 Single-cell versus Coupling Faults

Depending on the number of cells accessed during a SOS, the faults can be divided into single-cell and multi-cell faults (coupling faults). A single-cell fault occurs if only a single cell is involved into the SOS, whereas a coupling fault occurs if two or more cells are involved into the SOS. So, if a fault appears in the same cell, which the SOS is applied to, it is called single-cell fault; while, if the cell that sensitizes the fault is different from that where the fault appears, it is called a coupling fault.

In Fig. 2.1 also neighborhood pattern sensitivity faults (NPSFs) are listed which are a special type of coupling faults, where the states of the cells in the neighborhood of the victim cell influence the faulty behavior of the victim cell.

2.2.4 Address Decoder Faults

Address decoder faults (AFs) are caused by defects in the address decoder or in bit-lines and word-lines. There are four types of faults that concern the accessibility of memory cells [1]. In Fig 2.3, the four types of address decoder faults are illustrated.

- *No Access address fault (AFna)*: No cell is accessed with a certain address. Address A_x does not access cell C_x.
- *Multiple cell address fault (AFmc)*: One address accesses multiple cells. Address A_y accesses cells C_x and C_y.
- *Multiple address fault (AFma)*: One cell is accessed by two addresses. Addresses A_x and A_y access cell C_x.
- *Other cells address fault (AFoc)*: A certain address accesses multiple cells and one cell is accessed by multiple addresses. Address A_x and A_y access cell C_x, and Address A_y accesses cells C_x and C_y.

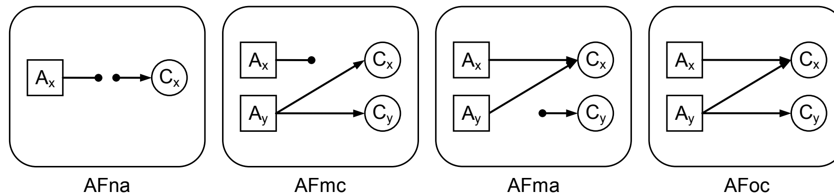

Figure 2.3. Address decoder faults

2.2.5 Peripheral Faults

Peripheral faults are caused by defects of peripheral read and write logic of the memory. Peripheral faults are [13]:

- Slow Write Driver Fault (SWDF)
- Slow Sense Amplifier Fault (SSAF)
- Slow Pre-charge Circuit Fault (SPRF)
- Bit-Line Imbalance Fault (BLIF)

Peripheral faults are not taken into account in the scope of this project.

2.3 Fault Primitives and Functional Fault Models

The topics of fault primitives and functional fault models are completely considered and described by Said Hamdioui in [2]. To introduce the faults in this work, Tables 2.1 and 2.2 summarize single-cell and coupling faults. A detailed description and explanation of these types of faults can be found in [2].

The nomenclature of single-cell FPs is <S/F/R>, where S is the state or operation sensitizing the fault. E.g. 0r0 means that the cell is in state '0' and a read operation is performed, where '0' is the expected value, or 0w1 means that the cell is in state '0' and a write 1 operation is performed. F denotes the faulty value of the failing cell, and R describes the logic output value of a read operation.

Table 2.1. Single-cell FPs and FFMs

#	FFM	FPs
1	SF	<1/0/->, <0/1/->
2	TF	<0w1/0/->, <1w0/1/->
3	WDF	<0w0/1/->, <1w1/0/->
4	RDF	<0r0/1/1>, <1r1/0/0>
5	DRDF	<0r0/1/0>, <1r1/0/1>
6	RRDF	<0r0/1/?>, <1r1/0/?>
7	IRF	<0r0/0/1>, <1r1/1/0>
8	RRF	<0r0/0/?>, <1r1/1/?>
9	USF	<1/?/->, <0/?/->
10	UWF	<0w0/?/->, <0w1/?/->, <1w0/?/->, <1w1/?/->
11	URF	<0r0/?/0>, <0r0/?/1>, <0r0/?/?>, <1r1/?/1>, <1r1/?/0>, <1r1/?/?>
12	SAF	<1/0/->, <0w1/0/->, <1w1/0/->, <0/1/->, <1w0/1->, <0w0/1/->
13	NAF	<0w1/0/->, <1w0/1/->, <0r0/0/?>, <1r1/1/?>
14	DRF	<0_T/1/->, <1_T/0/->, <0_T/?/->, <1_T/?/->

Table 2.1 summarizes 14 functional fault models and 28 fault primitives for single-cell faults. These FFMs and FPs are defined in detail in [2] and can be summarized as follows.

- *State Faults (SF)*: The value of the cell flips without any sensitizing operation and depends on the initial state of the cell.
- *Transition Faults (TF)*: The cell fails to flip when it is written with the opposite value. I.e. transition '0' → '1' or '1' → '0'.
- *Write Destructive Fault (WDF)*: A non-transition write operation (0w0 or 1w1) causes a transition.
- *Read Destructive Fault (RDF)*: A read operation causes the cell to flip and the incorrect value is returned to the output.
- *Deceptive Read Destructive Fault (DRDF)*: A read operation causes the cell to change its value, however the correct output is returned.
- *Random Read Destructive Faults (RRDF)*: A read operation flips the cell and a random logic value is returned to the output.
- *Incorrect Read Fault (IRF)*: A read operation returns the incorrect value to the output; however the stored value in the cell remains correct.
- *Random Read Fault (RRF)*: A read operation returns a random logic value to the output while the stored value remains correct.
- *Undefined State Fault (USF)*: Without any sensitizing operation, the logic value of a cell flips into an undefined state.
- *Undefined Write Fault (UWF)*: An undefined state of the cell is causes by a write operation.
- *Undefined Read Fault (URF)*: The cell is brought into an undefined state by a read operation.
- *Stuck-At Fault (SAF)*: The cell remains stuck at a value for any operation.
- *No Access Fault (NAF)*: The cell cannot be accessed. A write operation cannot change the value of the cell and a read

operation returns a random value. A NAF needs not to be caused by the address decoder, but can also be caused by an open word-line.

- *Data Retention Fault (DRF)*: The value of a cell changes after a certain time T without accessing the cell. (E.g. <0$_T$/1/-> denotes that the initial state of the cell is '0' and flips to '1' after a time T.)

For two-cell FPs, the nomenclature looks like <S_a;S_v/F/R>, where S_a is the state of sensitizing operation of the aggressor cell and S_v is state of sensitizing operation of the victim cell. F and R are the same as for single-cell FPs.

Table 2.2. Two-cell FPs and FFMs

#	FFM	FPs
1	CFst	<0;0/1/->, <0;1/0/->, <1;0/1/->, <1;1/0/->
2	CFus	<0;0/?/->, <0;1/?/->, <1;0/?/->, <1;1/?/->
3	CFds	<xwy;0/1/->, <xwy;1/0/->, <rx;0/1/->, <rx;1/0/->
4	CFud	<xwy;0/?/->, <xwy;1/?/->, <rx;0/?/->, <rx;1/?/->
5	CFid	<0w1;0/1/->, <0w1;1/0/->, <1w0;0/1/->, <1w0;1/0/->
6	CFin	{<0w1;0/1/->, <0w1;1/0/->}, {<1w0;0/1/->, <1w0;1/0/->}
7	CFtr	<0;0w1/0/->, <1;0w1/0/->, <0;1w0/1/->, <1;1w0/1/->
8	CFwd	<0;0w0/1/->, <1;0w0/1/->, <0;1w1/0/->, <1;1w1/0/->
9	CFrd	<0;0r0/1/1>, <1;0r0/1/1>, <0;1r1/0/0>, <1;1r1/0/0>
10	CFdrd	<0;0r0/1/0>, <1;0r0/1/0>, <0;1r1/0/1>, <1;1r1/0/1>
11	CFrrd	<0;0r0/1/?>, <1;0r0/1/?>, <0;1r1/0/?>, <1;1r1/0/?>
12	CFir	<0;0r0/0/1>, <1;0r0/0/1>, <0;1r1/1/0>, <1;1r1/1/0>
13	CFrr	<0;0r0/0/?>, <1;0r0/0/?>, <0;1r1/1/?>, <1;1r1/1/?>
14	CFuw	<x;0w0/?/->, <x;0w1/?/->, <x;1w0/?/->, <x;1w1/?/->
15	CFur	<x;0r0/?/0>, <x;0r0/?/1>, <x;0r0/?/?>, <x;1r1/?/0>, <x;1r1/?/1>, <x;1r1/?/?>

Table 2.2 summarizes the set of FFMs and FPs for coupling faults. A detailed definition can be found in [2]. The following items are a short summary.

- *State coupling fault (CFst)*: The v-cell is forced into a given logic state if the a-cell is in a given logic state without performing any operation on the v-cell or a-cell.
- *Undefined State coupling fault (CFus)*: The state of the v-cell is undefined while the a-cell is in a given logic state without performing any operation on the v-cell or a-cell.
- *Disturb coupling fault (CFds)*: Any operation performed on the a-cell causes the v-cell to flip.
- *Undefined Disturb coupling fault (CFud)*: Any operation performed on the a-cell forces the v-cell into an undefined state.
- *Idempotent coupling fault (CFid)*: A transition write operation on the a-cell causes the v-cell to flip.
- *Inversion coupling fault (CFin)*: A transition write operation on the a-cell inverts the logic value of the v-cell. CFin consists of two pairs of FPs that have to be present simultaneously. (Denoted by { and } in the notation of the FPs.)
- *Transition coupling fault (CFtr)*: A given logic value in the a-cell causes a failing transition write operation performed on the v-cell.
- *Write Destructive coupling fault (CFwd)*: A given logic state of the a-cell causes a transition in the v-cell although a non-transition write operation is performed on the v-cell.
- *Read Destructive coupling fault (CFrd)*: If the a-cell is in a given state, a read operation on the v-cell changes its value and returns the incorrect value to the output.
- *Deceptive Read Destructive coupling fault (CFdrd)*: If the a-cell is in a given state, a read operation on the v-cell changes its value and the correct value is returned to the output.
- *Random Read Destructive coupling fault (CFrrd)*: If the a-cell is in a given state, a read operation on the v-cell changes the value in the v-cell and a random value is returned to the output.

- *Incorrect Read coupling fault (CFir)*: If the a-cell is in a given state, a read operation on the v-cell returns the incorrect value to the output.
- *Random Read coupling fault (CFrr)*: If the a-cell is in a given state, a read operation on the v-cell returns a random value to the output while the value of the v-cell remains correct.
- *Undefined Write coupling fault (CFuw)*: A write operation on the v-cell forces it into an undefined state, while the a-cell is in a given state.
- *Undefined Read coupling fault (CFur)*: A read operation on the v-cell forces it into an undefined state, while the a-cell is in a given state. The value returned to the output can be correct, incorrect or random.

Chapter 3
Memory Test Algorithms

Testing memories in order to detect all different fault primitives, tests are used that perform a specific algorithmic sequence of read and write operation, i.e. a specific sensitizing operation sequence. During algorithmic memory testing the test sequence is applied sequentially to all addresses of a memory and hence, the whole memory is tested evenly.

In this chapter, the definition and structure of memory test algorithms are explained and algorithmic and environmental test parameters are described that are used in combination with memory test algorithms to improve the performance of the tests.

3.1 Nomenclature

The sensitizing operation sequence (SOS), i.e. the sequence of read and write operation that activates and detects the fault, has to be defined for algorithmic memory testing. Therefore an open notation for memory tests [14] has been developed that describes the SOSs and their usage in march elements of test algorithms.

Test algorithms consist of a sequence of march elements. The read and write operation of one march element are sequentially applied to one memory cell before moving to the next address, and one march element is applied to all addresses of a memory before moving to the next march element. For evaluating march algorithms, single march elements can be identified by numbering M_n. The numbering of march elements starts at n = 0; i.e. the first march element of an algorithm is M_0.

A test algorithm is delimited by curly brackets { and }, and each march element is delimited by parentheses (and). March elements are separated by a semicolon and the single read and write operation within a march element are separated by comas. An ⇩, ⇧, or ⇳ prior to the march elements denotes the addressing direction up (lowest address to highest), down (highest address to lowest) or arbitrary, respectively. An operation applied to a cell can be a ‘w0’ (write ‘0’), ‘r0’ (read ‘0’), ‘w1’ or ‘r1’. A *D* in the notation for March G denotes delay time between two march elements. *D* depends on factors like technology and clock frequency and is in a range of µs to seconds. For all tests in the frame of this work, *D* is set to 100ms.

The biggest part of this work refers to this nomenclature; for the use of other complex and more-dimensional test algorithms, additional symbols and nomenclature may be used. A list of additional symbols is provided in Table 3.1.

Table 3.1 Symbols and Nomenclature

Symbol	Meaning
⇧	address increment
⇩	address decrement
⇳	don't care address direction
↗	address increment along main diagonal
◆	N-E-S-W addressing around base cell
✸	N-NE-E-SE-S-SW-W-NW addressing around base cell
D	delay time for detecting data retention faults
b	apply to base cell
-b	apply to all cells except the base cell
Rep	apply the operation k times to n-cells with a distance of 2^k to the N, E, S & W of the b-cell
R-b	address row of base cell
C-b	address column of base cell
x	fast-x addressing (fast-row)
y	fast-y addressing (fast-column)

3.2 Test Algorithms

During development and definition of new fault primitives and functional fault models, the space of corresponding memory test algorithms also grows. Depending on the SOS, one or more faults models can be detected. In the beginning of the development of test algorithms during 1970's and 1980's, only few simple fault models like single-cell and static faults have been defined and hence also a few and simple test algorithms have been used. For example SCAN [1], MATS [1, 15, 16], March A [1, 17], March B [1, 17] or Algorithm B [18] are such traditional test algorithms. With the occurrence and definition of new and complex faults like linked and dynamic faults, new and specific test algorithms are needed. Based on the definition of specific FFMs, algorithms like March U [4], March LR [19], March RAW [20] or March AB1 [8] have been developed.

Much has been published on memory test algorithms, and a widespread literature survey has been done to provide a comprehensive list of test algorithms to select a subset of algorithms for the study. As a result of the literature survey, a

list of 51 memory test algorithms could be determined which is provided in Table 3.2. The table also includes algorithms that have been developed with respect to the properties of MBISTPLUS (e.g. hammering or random algorithms). These algorithms are Hammer5R, Hammer5W and Ham_Walk. MBISTPLUS is the embedded self-test used at Infineon and will be described in chapter 4.2.

Hammering means that the same read or write operation is repeatedly and sequentially performed during one march element. E.g. for Ham5R there are five sequential read operations in a march element (… ⇧(w0, r0, r0, r0, r0, r0) …).

Table 3.2. Memory test algorithms

#	Algorithm	Sequence	Reference
1.	SCAN	{⇳(w0);⇳(r0);⇳(w1);⇳(r1)}	[1, 21]
2.	SCAN+	{⇧(w0);⇧(r0);⇧(w1);⇧(r1);⇩(w0);⇩(r0);⇩(w1);⇩(r1)}	
3.	MATS	{⇳(w0);⇳(r0,w1);⇳(r1)}	[1]
4.	MATS+	{⇳(w0);⇧(r0,w1);⇩(r1,w0)}	[1, 16, 21]
5.	MATS++	{⇳(w0);⇧(r0,w1);⇩(r1,w0,r0)}	[1, 16, 22]
6.	March C-	{⇳(w0);⇧(r0,w1);⇧(r1,w0);⇩(r0,w1);⇩(r1,w0);⇳(r0)}	[1, 21, 22]
7.	March C--	{⇳(w0);⇧(r0,w1);⇧(r1,w0);⇩(r0,w1);⇩(r1,w0)}	[21, 22]
8.	March A	{⇳(w0);⇧(r0,w1,w0,w1);⇧(r1,w0,w1);⇩(r1,w0,w1,w0);⇩(r0,w1,w0)}	[1, 17]
9.	March B	{⇳(w0);⇧(r0,w1,r1,w0,r0,w1);⇧(r1,w0,w1);⇩(r1,w0,w1,w0);⇩(r0,w1,w0)}	[1, 17, 21]
10.	Algorithm B	{⇳(w0);⇧(r0,w1,w0,w1);⇧(r1,w0,r0,w1);⇩(r1,w0,w1,w0);⇩(r0,w1,r1,w0)}	[18, 22]
11.	March C+	{⇧(w0);⇧(r0,w1,r1);⇧(r1,w0,r0);⇩(r0,w1,r1);⇩(r1,w0,r0); ⇩(r0)}	[1]
12.	PMOVI	{⇩(w0);⇧(r0,w1,r1);⇧(r1,w0,r0);⇩(r0,w1,r1);⇩(r1,w0,r0)}	[22, 23]
13.	March 1/0	{⇧(w0);⇧(r0,w1,r1);⇩(r1,w0,r0);⇧(w1);⇧(r1,w0,r0); ⇩(r0,w1,r1)}	[24, 25]
14.	March TP	{⇩(w0);⇧(r0,w1);⇧(r1,w0);⇩(r0,w1,r1);⇩(r1,w0,r0)}	[22]
15.	March U	{⇳(w0);⇧(r0,w1,r1,w0);⇧(r0,w1);⇩(r1,w0,r0,w1);⇩(r1,w0);⇩(r0)}	[4]
16.	March X	{⇳(w0);⇧(r0,w1);⇩(r1,w0);⇳(r0)}	[1]
17.	March Y	{⇳(w0);⇧(r0,w1,r1);⇩(r1,w0,r0);⇩(r0)}	[1]
18.	March LR	{⇳(w0);⇩(r0,w1);⇧(r1,w0,r0,w1);⇧(r1,w0);⇧(r0,w1,r1,w0);⇧(r0)}	[19]
19.	March LA	{⇳(w0);⇧(r0,w1,w0,w1,r1);⇧(r1,w0,w1,w0,r0);⇩(r0,w1,w0,w1,r1); ⇩(r1,w0,w1,w0,r0);⇩(r0)}	[26]
20.	March RAW	{⇳(w0);⇧(r0,w0,r0,r0,w1,r1);⇧(r1,w1,r1,r1,w0,r0);⇩(r0,w0,r0,r0,w1,r1); ⇩(r1,w1,r1,r1,w0,r0);⇳(r0)}	[20, 7]
21.	March RAW1	{⇳(w0);⇳(w0,r0);⇳(r0);⇳(w1,r1);⇳(r1);⇳(w1,r1);⇳(r1);⇳(w0,r0);⇳(r0)}	[20, 7]
22.	March AB	{⇳(w1);⇩(r1,w0,r0,w0,r0);⇩(r0,w1,r1,w1,r1);⇧(r1,w0,r0,w0,r0); ⇧(r0,w1,r1,w1,r1);⇳(r1)}	[8, 27]
23.	March AB1	{⇳(w0);⇳(w1,r1,w1,r1,r1);⇳(w0,r0,w0,r0,r0)}	[8]
24.	March BDN	{⇧(w0);⇩(r0,w1,r1,w1,r1);⇩(r1,w0,r0,w0,r0);⇧(r0,w1,r1,w1,r1); ⇧(r1,w0,r0,w0,r0);⇧(r0)}	[28]
25.	March SR	{⇩(w0);⇧(r0,w1,r1,w0);⇧(r0,r0);⇧(w1);⇩(r1,w0,r0,w1);⇩(r1,r1)}	[29]
26.	March SR+	{⇩(w0);⇧(r0,r0,w1,r1,r1,w0,r0); ⇩(r0);⇧(w1);⇩(r1,r1,w0,r0,r0,w1,r1);⇧(r1)}	[2, 29]
27.	March SRD+	{⇩(w0);⇧(r0,r0,w1,r1,r1,w0,r0);***D***; ⇩(r0);⇧(w1);⇩(r1,r1,w0,r0,r0,w1,r1);***D***;⇧(r1)}	[2]
28.	March SS	{⇳(w0);⇧(r0,r0,w0,r0,w1);⇧(r1,r1,w1,r1,w0);⇩(r0,r0,w0,r0,w1); ⇩(r1,r1,w1,r1,w0);⇳(r0)}	[9]
29.	March SL	{⇳(w0);⇧(r0,r0,w1,w1,r1,r1,w0,w0,r0,w1);⇧(r1,r1,w0,w0,r0,r0,w1,w1,r1,w0); ⇩(r0,r0,w1,w1,r1,r1,w0,w0,r0,w1);⇩(r1,r1,w0,w0,r0,r0,w1,w1,r1,w0)}	[30]
30.	March G	{⇳(w0);⇧(r0,w1,r1,w0,r0,w1);⇧(r1,w0,w1);⇩(r1,w0,w1,w0);⇩(r0,w1,w0); ***D***;⇧(r0,w1,r1);***D***;⇧(r1,w0,r0)}	[31]
31.	GAL5R	{⇧(w0);⇧$_b$(w1$_b$,◆(r0,r1$_b$),w0$_b$);⇧(w1);⇧$_b$(w0$_b$,◆(r1,r0$_b$),w1$_b$)}	[21]
32.	GAL9R	{⇧(w0);⇧$_b$(w1$_b$,✸(r0,r1$_b$),w0$_b$);⇧(w1);⇧$_b$(w0$_b$,✸(r1,r0$_b$),w1$_b$)}	[21]
33.	GAL5W	{⇧(w0);⇧$_b$(w1$_b$,◆(w0,r1$_b$),w0$_b$);⇧(w1);⇧$_b$(w0$_b$,◆(w1,r0$_b$),w1$_b$)}	[21]
34.	GAL9W	{⇧(w0);⇧$_b$(w1$_b$,✸(w0,r1$_b$),w0$_b$);⇧(w1);⇧$_b$(w0$_b$,✸(w1,r0$_b$),w1$_b$)}	[21]
35.	Walking 1/0	{⇧(w0);⇧$_b$(w1$_b$,⇧$_{-b}$(r0),r1$_b$,w0$_b$);⇧(w1);⇧$_b$(w0$_b$,⇧$_{-b}$(r1),r0$_b$,w1$_b$)}	[1]
36.	Butterfly	{⇧(w0);⇧$_b$(w1$_b$,⇧$_{Rep}$(◆(r0),r1$_b$),w0$_b$);⇧(w1);⇧$_b$(w0$_b$,,⇧$_{Rep}$(◆(r1),r0$_b$),w1$_b$)}	[21]
37.	GALPAT	{⇳(w0);⇧$_b$(w1$_b$,⇧$_{-b}$(r0,r1$_b$),w0$_b$);⇳(w1);⇧$_b$(w0$_b$,⇧$_{-b}$(r1,r0$_b$),w1$_b$)}	[1]
38.	GALRow	{⇳(w0);⇧$_b$(w1$_b$,⇧$_{R-b}$(r0,r1$_b$),w0$_b$);⇳(w1);⇧$_b$(w0$_b$,⇧$_{R-b}$(r1,r0$_b$),w1$_b$)}	[1, 21]
39.	GALCol	{⇳(w0);⇧$_b$(w1$_b$,⇧$_{C-b}$(r0,r1$_b$),w0$_b$);⇳(w1);⇧$_b$(w0$_b$,⇧$_{C-b}$(r1,r0$_b$),w1$_b$)}	[1, 21]
40.	BLIF	{⇳(w0);$_x$⇳(w1,r1,w0);⇳(w1);$_x$⇳(w0,r0,w1)}	[22]
41.	HamW16	{⇧(w0);⇧(r0,w1^{16},r1);⇧(r1,w0^{16},r0);⇩(r0,w1^{16},r1);⇩(r1,w0^{16},r0)}	[22]
42.	HamR16	{⇧(w0);⇧(r0,w1,r1^{16},r1);⇧(r1,w0,r0^{16},r0);⇩(r0,w1,r1^{16},r1);⇩(r1,w0,r0^{16},r0)}	[22]
43.	HamR28	{⇧(w0);⇧(w1,w0,r0^{28});(w0,w1,r1^{28});⇩(w1,w0,r0^{28}); ⇩(w0,w1,r1^{28})}	[32]
44.	Ham_Max	{⇧(w1^{4},w0^{4},r0^{28});⇧(w0^{4},w1^{4},r1^{28})}	[32]
45.	Hammer_L	{⇧(w0^{8});⇧(r0^{28});⇧(w1^{8});⇧(r1^{28})}	[32]
46.	Hammer	{⇧(w0);↗(w1$_b^{1000}$,⇧$_{R-b}$(r0),r1$_b$,⇧$_{C-b}$(r0),r1$_b$,w0$_b$);⇧(w1); ↗(w0$_b^{1000}$,⇧$_{R-b}$(r1),r0$_b$,⇧$_{C-b}$(r1),r0$_b$,w1$_b$)}	[22]
47.	HamW	{⇧(w0);↗(w1$_b^{16}$,⇧$_{C-b}$(r0),w0$_b$);⇧(w1);↗(w0$_b^{16}$,⇧$_{C-b}$(r1),w1$_b$)}	[22]
48.	Ham5R	{⇧(w0);⇧(w1,r1^{5});⇧(w0,r0^{5});⇩(w1,r1^{5});⇩(w0,r0^{5})}	[32]
49.	Ham5W	{⇧(w0);⇧(w0^{5},r0);⇧(w1^{5},r1);⇩(w0^{5},r0);⇩(w1^{5},r1)}	[32]
50.	Ham_Walk	{⇧(w1);⇧(w0);⇧(r0,w1,r1,w0,r0);⇧(r0,w1);⇧(r1,w0,r0,w1,r1);⇧(r1)}	
51.	Random	{?(w?,r?)}	[32]

Note that the list of memory test algorithms cannot be assumed to be complete; however the most common algorithms are included.

Two types of test algorithms can be distinguished concerning addressing sequence: one-dimensional and more-dimensional algorithms. One-dimensional algorithms access one cell after another (marching algorithms), while more-dimensional algorithms are hopping through the cell array (galloping pattern), e.g. GALPAT. Marching algorithms are predominantly data oriented test pattern, and galloping patterns are primarily address oriented [10]. A galloping pattern performs a typical "ping-pong action" between one base cell and each other cell. More-dimensional algorithms are complex and more difficult to realize as often two nested address counters are needed, but the expected fault coverage of those pattern is expected to be very high [10]. In scope of this analysis, only one-dimensional algorithms could be taken into account, as the embedded BIST which is used for the study only supported one-dimensional march tests.

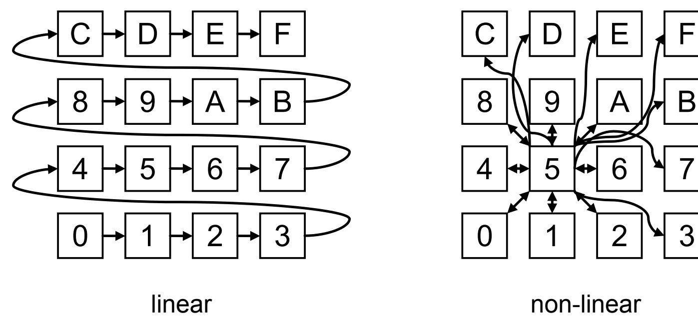

Figure 3.1. Addressing of one- and more-dimensional test algorithms

The examples given in Fig. 3.1 are showing a one-dimensional and a more-dimensional addressing sequence. The one-dimensional algorithm accesses one cell after the other, the more-dimensional example shows a GALPAT test

algorithm with cell 5 as base cell. At first the base cell is accessed by a write or read operation always prior to any other cell. The addressing sequence in this example is: “5→0→5→1→5→2→… …→5→F→5”. Two independent, nested addressing sequences are used that make GALPAT complex. GALPAT is a 2-dimensional test algorithm.

The sequence of one- or more-dimensional addressing is given by the algorithm itself and is part of the SOS.

3.3 Algorithmic Test Parameters

Additional to the SOS of memory test algorithms, additional algorithmic test parameters are used to influence the test performance and improve the detectability of faults [5]. Algorithmic Test Parameters are directly linked to the performance of test algorithms. They influence the addressing and test data of the algorithm. These parameters are:

- address direction
- addressing mode
- data background

All of these algorithmic parameters can be combined independently of each other.

3.3.1 Address Direction

The address direction denotes the order of incrementing and decrementing the row and column address. Either the row address or column address can be incremented (decremented) first. Hence, there are two types of address direction.

Fast-x addressing increments (decrements) the row address first [33]. Fast-x is also known as fast-row because the row address changes faster than the column address.

Fast-y addressing increments (decrements) the column address first. It is also named fast-column because each step goes to the next column [33].

In Fig.3.2 ,the address directions of fast-x and fast-y are illustrated at a 4x4 memory array with addresses 0 to F.

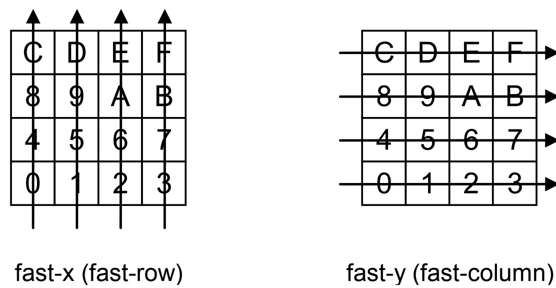

Figure 3.2. Addressing directions

3.3.2 Addressing Mode

The addressing mode defines the counting sequence of addressing the memory cells. Addressing modes are:

- linear
- 2^i (power of i)
- Grey code
- address complement

The addressing sequences of different addressing modes are illustrated in Fig. 3.3. The physical address layout for each example corresponds to the linear addressing mode. The numbering of the cell is showing the sequence.

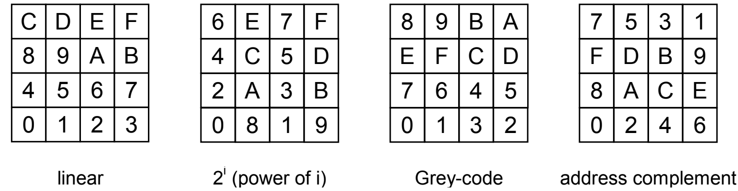

Figure 3.3. Address modes

Linear addressing accesses the memory cells along a row or column linearly one after another. This addressing mode is easy to realize and often used. The more complex addressing modes change the order of accessing the cells in specific ways.

Power of i (POI, 2^i) addressing [34, 35] accesses the 2nd (2^1), 4th (2^2), 8th (2^3) … cell next, depending on i. In Fig. 3.3, 2^i addressing is illustrated for i = 1. With POI, the position of the least significant bit (LSB) of the address can be varied. The exponent i denotes the position of the LSB in the address.

Grey code addressing uses the sequence of Grey to access the cells [36]. One property of Grey addressing is a Hamming distance of one; that means that only one bit of the address changes from one step to the next. Hence, this mode could be useful for testing on address decoder faults in asynchronous SRAMs. To achieve this, each single address bit needs to be checked separately to meet the internal timing constraints of the memory.

In contrast to Grey, address complement [34, 35] has the maximum hamming distance. With each step, all bits of the address are changed and additionally incremented (decremented) in each second step. So, each address is followed by its one's complement. The sequence for address complement shown in Fig. 3.3 is: $0 \rightarrow F \rightarrow 1 \rightarrow E \rightarrow \ldots$.

3.3.3 Data Background

The data background describes the pattern that inverts or non-inverts the data written to the memory. It can be described as a mask laying on the memory array. Data backgrounds are [21, 24, 36]:

- solid
- row-stripe
- column-stripe
- checkerboard

The illustration in Fig. 3.4 shows the pattern of non-inverted ('0') and inverted ('1') data for these four data backgrounds.

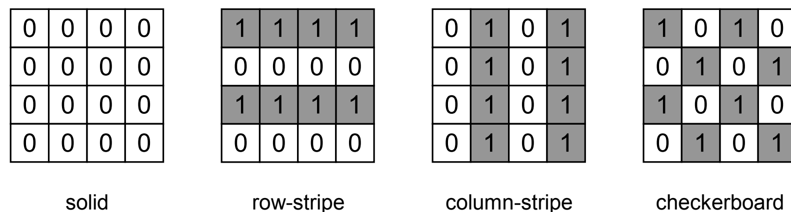

Figure 3.4. Data background patterns

These four data backgrounds are most regular and most common. However, any other pattern would also be possible. Due to a hardware bug in our tested memories, column-stripe and checkerboard could not be realized accurately. The background we used as column-stripe and checkerboard always combines two columns. The patterns are illustrated in Fig. 3.5.

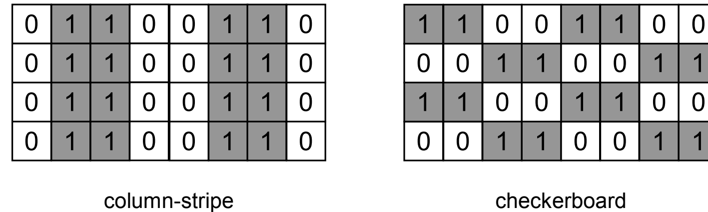

Figure 3.5. Buggy background patterns

3.4 Environmental Parameters

Additional to algorithmic parameters, environmental test parameters are influencing the test performance [37] and are used as stress parameters for the test. These parameters are given by the test environment and are:

- temperature
- supply voltage
- clock frequency

The environmental parameters are independent of the test algorithm and also of algorithmic parameters. They are also called non-algorithmic parameters. It depends on the test environment, which of the non-algorithmic stress parameters can be applied to the test. In the scope of this work, variations of temperature and supply voltage are taken into account. The frequency is not varied, but the tests are always performed at highest speed.

Chapter 4
Test Environment and Setup

The study is completely done during full productive memory testing. In this chapter, the basic techniques, software and hardware for memory testing are described. In particular the properties of the Infineon MBIST and the test strategy as a result of these properties are explained. Furthermore, the existing productive memory test flow is described where the study tests are included, and the way of data acquisition is explained as part of the test strategy.

4.1 Memory Testing

Two methods are well known for memory testing: MSIST (Memory Software-Implemented Self-Test, software based) and MBIST (Memory Built-In Self-Test, hardware based). Both are commonly used for productive memory tests and both have their specific advantages and disadvantages.

4.1.1 MSIST

In a software based memory test solution, a memory test program is executed via CPU [10] and the test patterns are written to the memory under test (MUT). The advantages of the so called Memory Software-Implemented Self-Test (MSIST) are high flexibility concerning test pattern update and easy implementation, as no additional logic is needed. However, the MUT needs to be accessible via CPU. Especially in large and complex SoCs smaller memories are eclipsed by logic and hence are not testable via MSIST. The program memory is also hardly testable in this way, because the read access may be limited and the write access may even be impossible. In contrast, the data memory is usually freely accessible by write and read operations.

A block diagram of a DUT containing several SRAMs is shown in Fig. 4.1. The test program is stored in a ROM and executed by the CPU on the SRAMs. Smaller SRAMs may not be accessible via CPU.

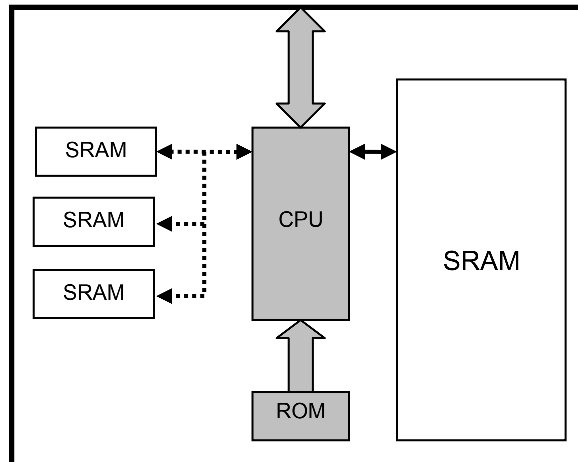

Figure 4.1. DUT block diagram using MSIST

4.1.2 MBIST

The hardware based Memory Built-In Self-Test (MBIST) is realized as an additional logic attached to each memory block, which controls the algorithmic test sequences on the memory cells. A MBIST can be implemented in different ways: micro-coded BIST or finite state machine BIST [10]. The BIST used for this study is a finite state machine (FSM) BIST. It basically consists of several registers and one or more finite state machines. Via the registers, the MBIST is configurable, while the test sequence is performed by the FSM. The advantages of MBIST are direct access to memories without CPU usage and the possibility of full speed testing. Also, even small memories that are not directly accessible via CPU can be tested via MBIST. A block diagram showing a device using MBISTs is given in Fig. 4.2.

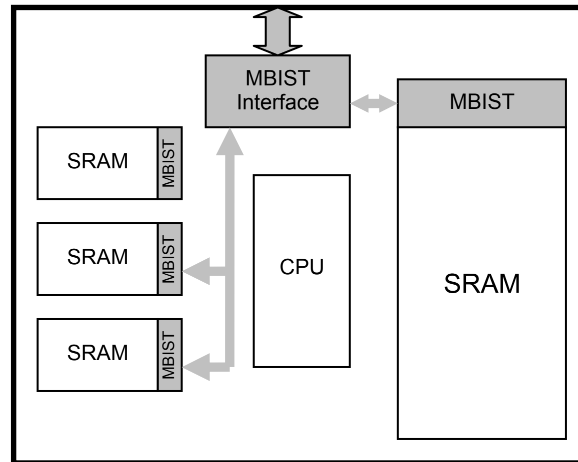

Figure 4.2. DUT block diagram using MBIST

Fig. 4.2 shows a device under test (DUT) with several SRAMs and MBISTs connected. One MBIST is attached to each SRAM, and the MBISTs are connected to the outside of the DUT to configure the test parameters and read the test results. The CPU is not in use for MBIST testing.

4.2 MBISTPLUS

At Infineon a proprietary design of MBIST called MBISTPLUS [32, 38] (previous: MBIST+) is used for automotive microcontroller devices. MBISTPLUS is a configurable BIST that is able to perform predefined tests but also allows to program own algorithmic march tests and select a couple of algorithmic test parameters. Registers are available to configure the test settings and to store the test results.

If no own parameters are set, the standard setting (RESET configuration) automatically performs the following tests:

- SCAN, linear, fast-y, row-stripe
- SCAN, linear, fast-y, column-stripe
- SCAN, linear, fast-y, solid
- SCAN, linear, fast-y, checkerboard
- March C+, linear, fast-y, solid

However, the tests of the RESET configuration are only a very small part of potentially possible tests. In the scope of this project, two generations of MBISTPLUS have been taken into account: MBISTPLUS V3.0 and MBISTPLUS V4.2. Both versions have been analyzed concerning their properties and potentially possible tests prior to implementing a study test set. These properties of MBISTPLUS are summarized in table 4.1.

Table 4.1. Properties of MBISTPLUS

Parameters	**MBISTPLUS V3.0**	**MBISTPLUS V4.2**
Algorithms	one-dimensional, marching (lengths of ME ≤ 6)	one-dimensional, marching (lengths of MEs ≤ 6) GAL5R, GAL9R, Hammer, Random
Addressing Mode	linear	linear, 2^i, Grey-Code, Address Complement
Address Direction	fast-x, fast-y	fast-x, fast-y
Data Background	solid, row-stripe, column-stripe, checkerboard	solid, row-stripe, column-stripe, checkerboard
Self Timing		read timing, write timing, weak write driver

Both versions of MBISTPLUS are able to perform one-dimensional marching algorithms where the length of single march elements is restricted to at most six operations. Furthermore, address directions fast-x and fast-y, as well as four data backgrounds solid, row-stripe, column-stripe and

checkerboard are supported. While MBISTPLUS V3.0 only supports linear addressing, MBISTPLUS V4.2 additionally contains the addressing modes POI, Grey-code and address complement. MBISTPLUS V4.2 also allows to perform predefined 2-dimensional algorithms GAL5R and GAL9R, and has special registers to configure explicit hammering and random tests.

As only one-dimensional march algorithms are supported as self configurable test algorithms, the possible number of different algorithms is restricted by this fact. Hence, a selection of 30 algorithms that can be used with MBISTPLUS is given in Table 4.2.

Table 4.2. Memory test algorithms

#	Algorithm	Sequence
1	SCAN	{⇳(w0);⇳(r0);⇳(w1);⇳(r1)}
2	SCAN+	{⇧(w0);⇧(r0);⇧(w1);⇧(r1);⇩(w0);⇩(r0);⇩(w1);⇩(r1)}
3	MATS	{⇳(w0);⇳(r0,w1);⇳(r1)}
4	MATS+	{⇳(w0);⇧(r0,w1);⇩(r1,w0)}
5	MATS++	{⇳(w0);⇧(r0,w1);⇩(r1,w0,r0)}
6	March C-	{⇳(w0);⇧(r0,w1);⇧(r1,w0);⇩(r0,w1);⇩(r1,w0);⇳(r0)}
7	March A	{⇳(w0);⇧(r0,w1,w0,w1);⇧(r1,w0,w1);⇩(r1,w0,w1,w0);⇩(r0,w1,w0)}
8	March B	{⇳(w0);⇧(r0,w1,r1,w0,r0,w1);⇧(r1,w0,w1);⇩(r1,w0,w1,w0);⇩(r0,w1,w0)}
9	Algorithm B	{⇳(w0);⇧(r0,w1,w0,w1);⇧(r1,w0,r0,w1);⇩(r1,w0,w1,w0);⇩(r0,w1,r1,w0)}
10	March C+	{⇧(w0);⇧(r0,w1,r1);⇧(r1,w0,r0);⇩(r0,w1,r1);⇩(r1,w0,r0); ⇩(r0)}
11	PMOVI	{⇩(w0);⇧(r0,w1,r1);⇧(r1,w0,r0);⇩(r0,w1,r1);⇩(r1,w0,r0)}
12	March 1/0	{⇧(w0);⇧(r0,w1,r1);⇩(r1,w0,r0);⇧(w1);⇧(r1,w0,r0); ⇩(r0,w1,r1)}
13	March TP	{⇩(w0);⇧(r0,w1);⇧(r1,w0);⇩(r0,w1,r1);⇩(r1,w0,r0)}
14	March U	{⇳(w0);⇧(r0,w1,r1,w0);⇧(r0,w1);⇩(r1,w0,r0,w1);⇩(r1,w0);⇩(r0)}
15	March X	{⇳(w0);⇧(r0,w1);⇩(r1,w0);⇳(r0)}
16	March Y	{⇳(w0);⇧(r0,w1,r1);⇩(r1,w0,r0);⇩(r0)}
17	March LR	{⇳(w0);⇩(r0,w1);⇧(r1,w0,r0,w1);⇧(r1,w0);⇧(r0,w1,r1,w0);⇧(r0)}
18	March LA	{⇳(w0);⇧(r0,w1,w0,w1,r1);⇧(r1,w0,w1,w0,r0);⇩(r0,w1,w0,w1,r1); ⇩(r1,w0,w1,w0,r0);⇩(r0)}
19	March RAW	{⇳(w0);⇧(r0,w0,r0,r0,w1,r1);⇧(r1,w1,r1,r1,w0,r0);⇩(r0,w0,r0,r0,w1,r1); ⇩(r1,w1,r1,r1,w0,r0);⇳(r0)}
20	March RAW1	{⇳(w0);⇳(w0,r0);⇳(r0);⇳(w1,r1);⇳(r1);⇳(w1,r1);⇳(r1);⇳(w0,r0);⇳(r0)}
21	March AB	{⇳(w1);⇩(r1,w0,r0,w0,r0);⇩(r0,w1,r1,w1,r1);⇧(r1,w0,r0,w0,r0); ⇧(r0,w1,r1,w1,r1);⇳(r1)}
22	March AB1	{⇳(w0);⇳(w1,r1,w1,r1,r1);⇳(w0,r0,w0,r0,r0)}
23	March BDN	{⇧(w0);⇩(r0,w1,r1,w1,r1);⇩(r1,w0,r0,w0,r0);⇧(r0,w1,r1,w1,r1); ⇧(r1,w0,r0,w0,r0);⇧(r0)}
24	March SR	{⇩(w0);⇧(r0,w1,r1,w0);⇧(r0,r0);⇧(w1);⇩(r1,w0,r0,w1);⇩(r1,r1)}
25	March SS	{⇳(w0);⇧(r0,r0,w0,r0,w1);⇧(r1,r1,w1,r1,w0);⇩(r0,r0,w0,r0,w1); ⇩(r1,r1,w1,r1,w0);⇳(r0)}

Table 4.2. Memory test algorithms (cont.)

#	Algorithm	Sequence
26	BLIF	{⇕(w0);$_x$⇕(w1,r1,w0);⇕(w1);$_x$⇕(w0,r0,w1)}
27	Ham5R	{⇧(w0);⇧(w1,r1^5);⇧(w0,r0^5);⇩(w1,r1^5);⇩(w0,r0^5)}
28	Ham5W	{⇧(w0);⇧(w0^5,r0);⇧(w1^5,r1);⇩(w0^5,r0);⇩(w1^5,r1)}
29	March G	{⇕(w0);⇧(r0,w1,r1,w0,r0,w1);⇧(r1,w0,w1);⇩(r1,w0,w1,w0);⇩(r0,w1,w0); ***D***;⇧(r0,w1,r1);***D***;⇧(r1,w0,r0)}
30	Ham_Walk	{⇧(w1);⇧(w0);⇧(r0,w1,r1,w0,r0);⇧(r0,w1);⇧(r1,w0,r0,w1,r1);⇧(r1)}

With the selection of test algorithms and the properties of MBISTPLUS, a specific number of different tests can be combined and executed by the two version of MBISTPLUS used in this project. To estimate the quality of MBISTPLUS, the potential of each version of MBISTPLUS is analyzed.

4.2.1 Potential of MBISTPLUS V3.0

With 30 algorithms of Table 4.2, and the properties listed in Table 4.1, the following parameters can be configured in MBISTPLUS V3.0:

- 30 one-dimensional test algorithms
- 1 addressing mode
- 2 addressing directions
- 4 data backgrounds

Hence, the total number of possibly configurable tests is:

$$30 \cdot 1 \cdot 2 \cdot 4 = 240 \tag{1}$$

Any of these possible tests could be combined to a test set. If any combination is considered, the maximum number of possible test sets is:

$$2^{240} \; (\approx 10^{72}) \tag{2}$$

Even though only four parameters are configurable in MBISTPLUS V3.0, an enormous number of possible tests and test sets are possible. This shows that the RESET configuration which is used by default is not even close to the potential of the MBIST.

4.2.2 Potential of MBISTPLUS V4.2

With MBISTPLUS V4.2 even more parameters can be configured than in MBISTPLUS V3.0. Especially the configurable self timing enables an immense number of additional possibilities. The self timing parameters read timing, write timing and weak write driver are configurable by a 14-bit register. The setting influences the timing in such way, that read and write switching times are shifted and corner cases become faulty. With 14 bit, a number of 2^{14} self timing settings are possible. Hence, the parameters of MBISTPLUS V4.2 are:

- 30 one-dimensional test algorithms
- 4 addressing modes
- 2 address directions
- 4 data backgrounds
- 2^{14} self timing configurations
- 4 additional sequencer tests (GAL5R, GAL9R, Hammer, Random)

The possible number of configurable tests is

$$\left(30 \cdot 4 \cdot 2 \cdot 4 \cdot 2^{14}\right) + 4 = 15728644 \tag{3}$$

and the theoretical possible number of test sets is

$$2^{15728644} \tag{4}$$

Both, the number of possible tests and test sets far exceed the possibilities of productive memory testing.

4.3 Burn-In

Burn-In [39] is part of quality assurance of memory devices to detect latent faulty devices that would fail in long-term usage. In Fig. 4.3 the well known bathtub curve [40] and its three phases of life time (infant mortality, useful life time, and wear-out) is shown.

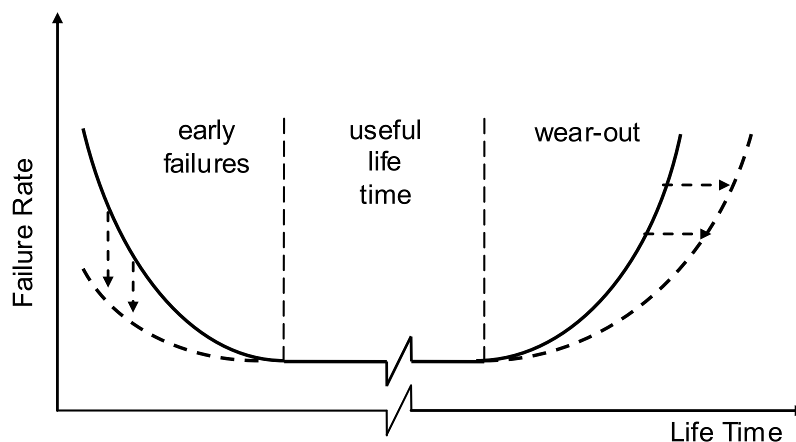

Figure 4.3. Bathtub curve

Aim of Burn-In is to keep the phase of early failures as short as possible, and delay the wear-out phase as much as possible to extend the phase of useful life time that has the minimum failure rate. During Burn-In the devices are exposed to high voltage and temperature stresses that cause artificial aging [1, 41]. Aging causes slowing down switching operations of transistors in the SRAM cell. Latent faults may be caused by weak transistors or marginal values of resistance

within a memory cell, causing setup and hold time violations [1]. Latent faults occur sporadically and need to be stabilized in order to be detected safely.

Faults that become detectable due to Burn-In are often caused by sensitive design or process variation [2]. Latent faults become detectable early and can be sorted out. Hence, the detection of latent faults then decreases the phase of early failures and delays the wear-out phase at the same time. However, the delay of wear-out is less important as the economical lifetime of an electronic device is often shorter [1]. In Fig. 4.3, the bathtub curve is shown, and the effects of Burn-In (broken line) are illustrated.

4.4 Test Strategy

The objectives of the project are to analyze the effectiveness of memory test algorithms and test set optimization. That means to combine effective test algorithms into test sets, such that the number of detected faults is as high as possible while test time is as low as possible at the same time.

The main strategy of the project is to use statistical analysis of productive test results to achieve knowledge about the faults that occur. In contrast to previous studies, the fault models are unknown in productive testing and the simple fail information of which algorithm detected a fault and which did not combined with the information about algorithmic and environmental test parameters is used to analyze the faulty behavior and to conclude possible functional fault models.

The test algorithms for the study have to be placed into the productive test flow of memories and the results have to be stored and provided for the analysis. In order to achieve a meaningful outcome of the study, the number of test results has to be representative. Hence, a large number of different test algorithms should be performed over a long period of productive testing to gather a sufficient number of test results.

Both, tests and data acquisition needed to be inserted into the industrial, full productive test flow.

The test strategy of the study is based on the so called "Kitchen-Sink-Principle" [6]. Any test should be applied to achieve the maximum number of test results; i.e. it is taken anything but the kitchen sink as the starting point of the analysis. Due to the fact that the maximal number of tests is applied at once, any analysis is possible. The application of a reduced set or independent sets would mean that not all relationships could be analyzed.

It is necessary to apply as many algorithms as possible to as many devices and memories as possible to achieve a meaningful statistical basis. In difference to theory, the fault models that appear during testing are unknown, and so the selection of efficient algorithms cannot be based on theoretical relations between fault models and test algorithms. This study used the experimental results as starting point to analyze the efficiency of test algorithms and to draw conclusions on possible fault models from the statistical analysis.

4.5 Test Setup

To achieve a sufficient number of test results for a meaningful statistical analysis, a study test set was placed into the productive test flow for embedded memories of microcontroller devices. For this analysis, a product was chosen that has been in full production, because a high throughput and hence a large number of test results could be expected.

4.5.1 Tested Devices and Memories

For the study, Infineon TC1797 automotive microcontroller devices have been used. The TC1797 controller is a 32-bit microcontroller of 130 nm technology which contains MBISTPLUS V3.0. For all tests, the maximum frequency of 180MHz was used. The productive throughput of those devices has been high enough to obtain enough test results. During the time, the test data have been gathered, hundreds of thousands of devices have been tested.

Each TC1797 controller contains thirteen embedded SRAMs of different size that were accessible via MBIST. The memories are sized between 1.38kB and 128kB and the total size of the tested memories is 261.56KB per device. Because of their size, instruction memory (PMI, 40kB) and data memory (DMI, 128kB) are most important for the study. A block diagram [42] of TC1797 is shown in Fig. 4.4.

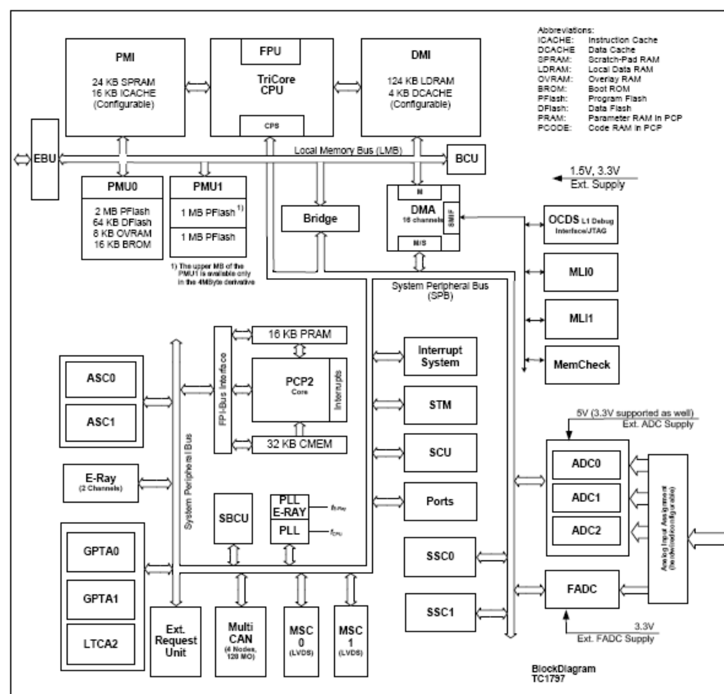

Figure 4.4. TC1797 block diagram

From the data given above, the necessary test time can be calculated according to formula (5), where memory_size (n) is 261.56kB, frequency (f) is 180MHz and test_length and delay_time (D) depend on the test set. This is the pure test time without any time for setup, configuration of MBISTPLUS or read and storing time.

$$TT = \frac{test_length \cdot memory_size}{frequency} + delay_time \quad (5)$$

4.5.2 Study Test Set

According to the properties and possibilities of MBISTPLUS V3.0, a large study test set was combined of test algorithms and algorithmic test parameters. The conditions of the study test set are summarized in Table 4.3.

Table 4.3. Study test set

#	Algorithm	Test Length	Algorithmic Parameter							
			fast-x (fx)				fast-y (fy)			
			so	rs	cs	cb	so	rs	cs	cb
1	SCAN	4n	+	+	+	+	+	+	+	+
2	SCAN+	8n	+	+	+	+	+	+	+	+
3	MATS	4n	+	+	+	+	+	+	+	+
4	MATS+	5n	+	+	+	+	+	+	+	+
5	MATS++	6n	+	+	+	+	+	+	+	+
6	March C-	10n	+	+	+	+	+	+	+	+
7	March A	15n	+	+	+	+	+	+	+	+
8	March B	17n	+	+	+	+	+	+	+	+
9	Algorithm B	17n	+	+	+	+	+	+	+	+
10	March C+	14n	+	+	+	+	+	+	+	+
11	PMOVI	13n	+	+	+	+	+	+	+	+
12	March 1/0	14n	+	+	+	+	+	+	+	+
13	March TP	11n	+	+	+	+	+	+	+	+
14	March U	14n	+	+	+	+	+	+	+	+
15	March X	6n	+	+	+	+	+	+	+	+

Table 4.3. Study test set (cont.)

#	Algorithm	Test Length	Algorithmic Parameter							
			fast-x (fx)				fast-y (fy)			
			so	rs	cs	cb	so	rs	cs	cb
16	March Y	8n	+	+	+	+	+	+	+	+
17	March LR	14n	+	+	+	+	+	+	+	+
18	March LA	22n	+	+	+	+	+	+	+	+
19	March RAW	26n	+	+	+	+	+	+	+	+
20	March RAW1	13n	+	+	+	+	+	+	+	+
21	March AB	22n	+	+	+	+	+	+	+	+
22	March AB1	11n	+	+	+	+	+	+	+	+
23	March BDN	22n	+	+	+	+	+	+	+	+
24	March SR	14n	+	+	+	+	+	+	+	+
25	March SS	22n	+	+	+	+	+	+	+	+
26	BLIF	8n	+	+	+	+	-	-	-	-
27	Ham5R	25n	+	-	-	-	+	-	-	-
28	Ham5W	25n	+	-	-	-	+	-	-	-
29	March G	23n+2*D*	+	+	+	+	+	+	+	+
30	Ham_Walk	15n	+	+	+	+	+	+	+	+

A '+' denotes that the algorithm was combined with the corresponding parameters, while a '-' denotes that the corresponding parameter was not applied. The algorithmic parameters are address direction (fast-x and fast-y), and four data backgrounds solid (so), row-stripe (rs), column-stripe (cs) and checkerboard (cb). The test length is the number of operation per algorithms and hence denotes the duration of a test in n.

Algorithm BLIF (#26) is only executed with fast-x addressing as it is especially designed for bit-line imbalance faults, and the two hammer tests Ham5R and Ham5W (#27 and #28) are only executed with solid data background.

In total, 224 different tests are performed by the study test set with a total length of $3092 \cdot \frac{n}{f} + 16D$ (test_length = 3092, delay_time = 16*D* with *D* = 100ms)). The test time for each cycle of the whole test set applied to all memories is then:

$$TT_{cycle} = \frac{3092 \cdot 261.56kB \cdot \frac{1024 \cdot 8}{32kB}}{180MHz} + 16 \cdot 100ms = 2.75s \quad (6)$$

Note that the memories are accessed wordwise, so, 32Bits are always accessed in parallel and the memory size that is used for calculating the test time is converted.

4.5.3 Test Environment

Embedded memories of a product are tested several times at different stages of production. One of these tests is the so called IBIS (Interconnect Built-In Self-Test) flow [43]. The IBIS flow is placed after wafer test and packaging (see Fig. 4.5). That means that only devices that already have passed wafer testing appear in this study. And the faults that can be detected either slipped wafer testing or came into existence after wafer test or packaging.

IBIS is an innovative test solution that integrates test and Burn-In in one system [44]. The DUTs are placed on a Burn-In board which is put into a Burn-In oven for a period of 12 hours. Via the Burn-In board, the DUTs are connected to the outside and hence are controllable and testable.

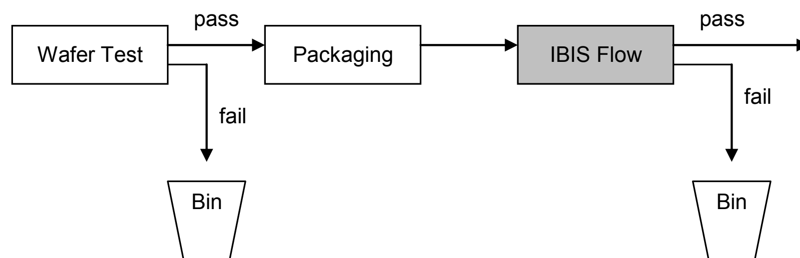

Figure 4.5. Memory test flow

4.5.4 Test Flow

The study test set was integrated into the IBIS flow and is performed seven times during the IBIS flow at different environmental conditions before and after Burn-In and high voltage stress [45]. The IBIS flow is part of the productive test plan and had to be taken as given for the project. The relevant part of the test flow is shown in Fig. 4.6. Each block of tests is identified by a test number (TN), where the same set of tests was applied, however at different environmental test conditions. The test numbers and corresponding environmental conditions temperature and supply voltage are given in Table 4.4 and Fig. 4.7.

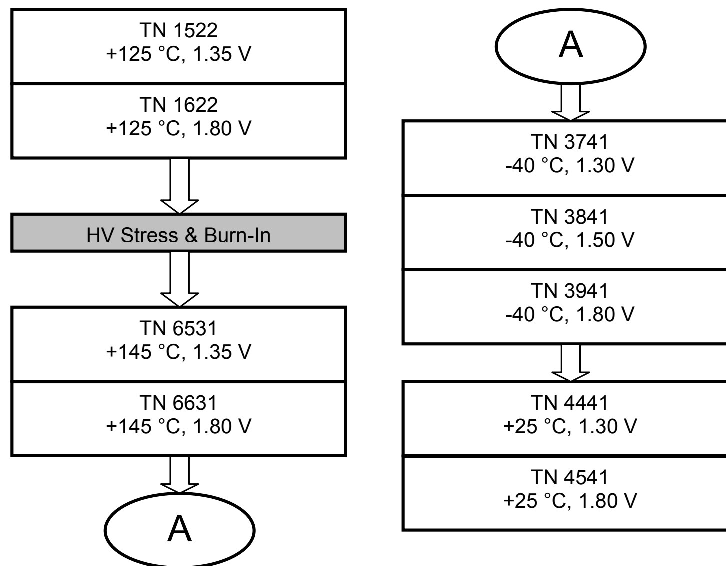

Figure 4.6. Test flow

Table 4.4. Test numbers and environmental conditions

Test Number	Temperature	Voltage
3741	-40°C	1.30V
3841	-40°C	1.50V
3941	-40°C	1.80V
4441	+25°C	1.30V
4541	+25°C	1.80V
1522	+125°C	1.35V
1622	+125°C	1.80V
6531	+145°C	1.35V
6631	+145°C	1.80V

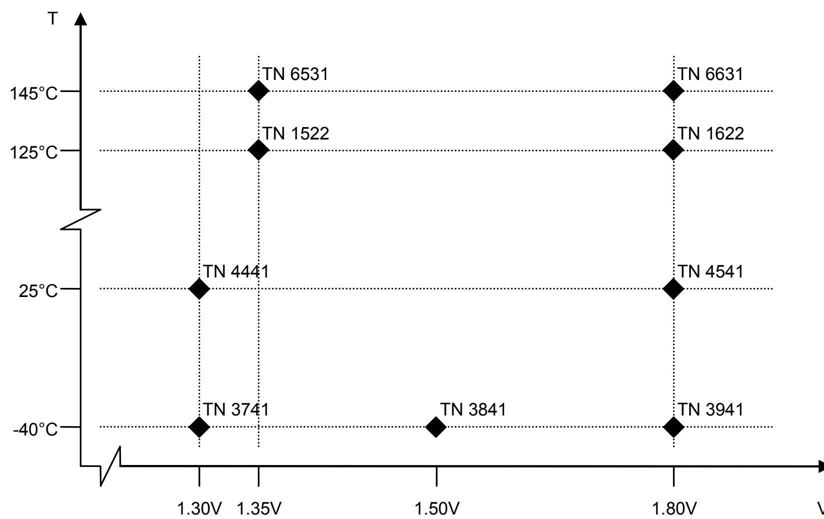

Figure 4.7 Test numbers and environmental conditions

There are only two tests cycles performed before Burn-In (TN 1522 and TN 1622), all other tests are performed after Burn-In.

The fact that the test set is performed repeatedly at different environmental conditions before and after Burn-In, allows to analyze the influence of those conditions on the test results. The test number is used to identify the different test conditions.

An important note is that all the tests of the study were included into the IBIS flow. That means that all faults that are recognized in this study are either affected by Burn-In or the high voltage stress during the test flow, or have slipped through wafer test.

4.5.5 Data Acquisition

IBIS flow as well as MBISTPLUS has not been designed for such a large data analysis as in this project. So, the comprehensive test result data have to be buffered several times. During the tests, the fail information of each single test algorithm that is executed is stored as a fail bit in a register of MBISTPLUS. To clear the register for the next cycle, all fail bits of the previous cycle have to be buffered on the device. For this purpose a previously tested and fault free memory was used. After all tests are finished, the whole buffered fail information is readout by the IBIS system and the raw data are written to a test result file (TRF). However, the information of the TRF is highly compressed.

For further analysis, the information has to be reprocessed and the test data are written to a SQL database for more comfortable handling. The SQL database then contains all fail information in combination with any information about test parameters (test number, environmental parameters and algorithmic parameters). Thus, each fault can exactly be identified and related to the conditions it occurred.

In Fig. 4.8, the process of testing and data acquisition is illustrated as a block diagram.

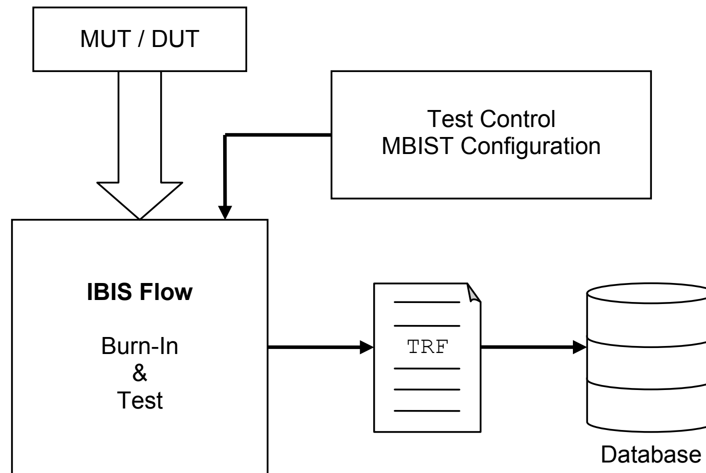

Figure 4.8. IBIS flow and data acquisition

The analysis of the test results is done by pure data mining of the information in the SQL database. Due to the detailed storage of data, the information can be combined to any query on relationship of algorithm, faults and parameters. The database does not contain any false positive results. False positive results could be caused by communication errors or test program errors that cause the test to fail, even though no memory error occurred. The pre-processing of the data ensures that only true faults (i.e. only fails caused by memory faults) are entered into the database.

Chapter 5
Fault Coverage of Test Algorithms

In productive memory testing the efficiency of test algorithms is essential. The productive test time has to be kept short and the fault coverage has to be as high as possible at the same time. Hence, the selection of tests for a productive test set highly depends on the efficiency of test algorithms and test sets.

In this chapter the performance of single algorithms, based on their fault coverage at different environmental conditions, and on the other hand also the efficiency of combinations of two algorithms is analyzed. Both results are used to classify the algorithms with similar properties concerning the coverage of specific fault models, and to estimate the distribution of those fault models within the test results.

5.1 Definitions

In this section, terms that are used for the analysis will be explained.

5.1.1 Fault Coverage

The fault coverage (FC) of a test or algorithm is the number of faults detected, related to the total number of faults. The fault coverage is then given as a percentage. The FC shows the amount of faults that a specific test or algorithm detects, and allows comparing the efficiency of single tests to each other. The higher the FC of a test or algorithm is, the more efficient it is.

The following aspects are considered for the analysis:

- fault coverage at specific environmental conditions
- fault coverage of single test algorithms (at constant environmental conditions)

5.1.2 Test

A test is defined as the combination of test algorithm (see. chapter 3.2), algorithmic (chapter 3.3) and environmental test parameters (chapter 3.4), where the environmental test parameters are given as test number (see Table 4.4). These are the minimum requirements to perform a memory test.

5.1.3 Test Set

A test set is a sequence of two or more tests performed as a group.

5.2 Fault Coverage at Different Environmental Conditions

The fault coverage of the test varies with environmental test conditions. In this analysis, the effects of temperature and supply voltage are analyzed. The different conditions can be identified by the test number. According to Table 4.4 and Fig. 4.7, for each test number (i.e. for each combination of temperature and supply voltage), the number of detected faults is determined; and also the number of faults that are exclusively detected at these test conditions. Hence, the fault coverage could be determined for each combination of environmental test conditions.

5.2.1 Test Results

The results of this analysis are given in Table 5.1. In total, 2712 faults are analyzed. This is the maximum number of faults detected by all tests. So, for the following analysis, it is assumed that 2712 faults refer to 100% fault coverage, although there may be additional faults that could not be detected by any test of the set. For each test number, and hence, environmental test condition and cumulated over all test algorithms, the number of faults (# of faults) is determined and also the number of those faults that are detected exclusively (# of faults excl.), i.e. only with these test conditions. Additional to test numbers, the environmental parameters supply voltage (V) and temperature (T), and also FC for each TN are included in Table 5.1.

Table 5.1. Fault coverage per test number

	before BI		after Burn-In						
TN	**1522**	**1622**	**6531**	**6631**	**3741**	**3841**	**3941**	**4441**	**4541**
V	1.35	1.80	1.35	1.80	1.30	1.50	1.80	1.30	1.80
T	+125	+125	+145	+145	-40	-40	-40	+25	+25
# of faults	617	56	2439	175	237	70	46	165	25
# of faults (excl.)	4	1	811	48	37	0	12	3	0
FC (total)	22,60%	2,03%	60,03%	4,68%	7,37%	2,58%	1,25%	5,97%	0,92%
FC (excl.)	0,15%	0,04%	29,90%	1,77%	1,36%	0,00%	0,44%	0,11%	0,00%

The fault coverage is graphically shown in Fig. 5.1, and the ratio of fault coverage and exclusive faults is given in Fig. 5.2. For both diagrams, 100% refers to the total number of 2712 faults.

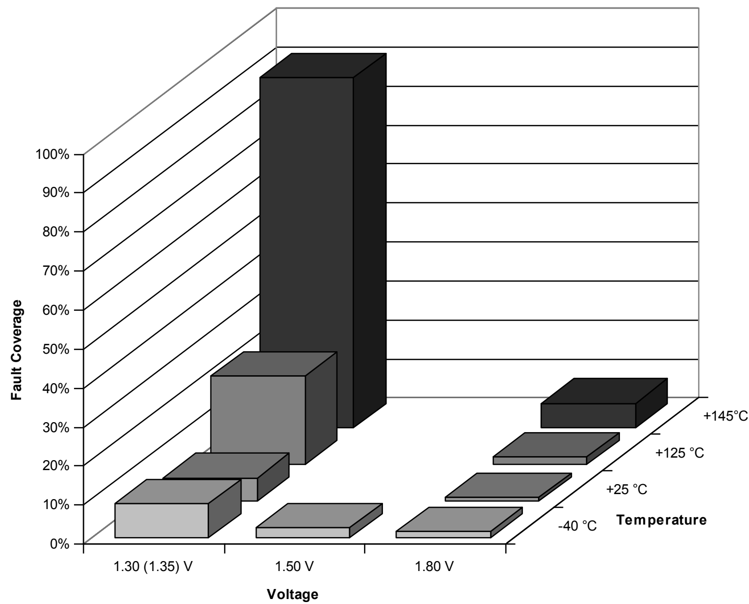

Figure 5.1. Fault coverage

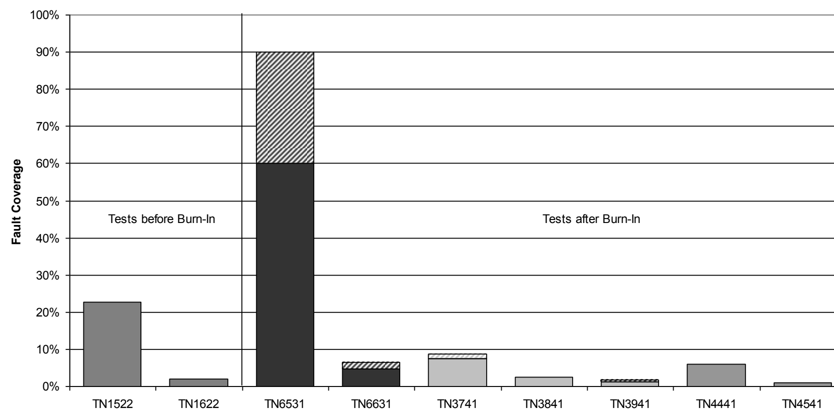

Figure 5.2. Fault coverage and exclusive faults

Fig. 5.2 shows the total fault coverage for each test number and the part of faults that are detected exclusively as hachured area.

5.2.2 Data Evaluation

The highest fault coverage can be observed at TN6531 (1.35V / +145°C). 2439 of 2712 faults are detected (89,9%), while the fault coverage of all other test number is significantly lower. The coverage of TN1522, which takes place at nearly the same environmental conditions (1,35V / +125°C) is only at 23%. In this case, the effect of Burn-In becomes visible that increases the fault coverage comparing the results of TN1522 before and TN6531 after Burn-In. For the same reason, the number of exclusive faults is highest at TN6531. However, the artificial aging dependent effects of Burn-In on the fault coverage will be analyzed and described in more detail in Chapter 8.

Here, the effects of temperature and supply voltage are of interest. Comparing the results shown in Fig. 5.1, one can see that the fault coverage increases with increasing temperature and decreasing supply voltage. However, the effect is much

more pronounced with increasing temperature. Both causes a relatively higher fault coverage in the corner of low supply voltage and high temperature, and a low fault coverage in the corner of high supply voltage and low temperature. These corner cases are also called “slow corner” and “fast corner” respectively, because “cell delay increases with decreasing voltage and increasing temperature” [46]. This is because the environmental conditions temperature and voltage influence the timing of transistors [41]. The increase of fault coverage in the slow corner is much more pronounced, while the effect of the fast corner is hardly visible. The voltage dependent effect could be expected as similar results have been shown in previous investigations [33].

An interesting fact is that, concerning the tests performed at low supply voltage, the fault coverage decreases with decreasing temperature as expected, but increases again a little at lowest temperature (supply voltage: 1.30V, temperature: -40°C).

In conclusion, the effects of temperature variation during tests are significantly recognizable, while the effects of supply voltage variation are lower in comparison. Nevertheless, both causes an increase of fault coverage in the slow corner, which means that the highest fault coverage is recognized at maximum temperature (+145°C) and low supply voltage (1.35V). High temperature testing is very effective to detect many faults and even exclusive faults. Nevertheless, at room and low temperature also exclusive faults are detected, so that testing at different environmental conditions is not avoidable.

5.3 Fault Coverage of Test Algorithms

Each test algorithm is able to detect a couple of functional fault models. The number of faults that is detected by an algorithm denotes its fault coverage. In this section the fault coverage of the single algorithms within the study test set are analyzed and the effectiveness of these algorithms is evaluated.

Due to the fact, that the test set is performed seven times at different environmental conditions, there are also seven different test results for the fault coverage of the test algorithms. The results of TN6531 are most meaningful; first, because this test gives the most results (2439 faults) and second, the test is performed after Burn-In and thus also includes the effects of Burn-In.

So, the results of TN6531 will be evaluated in this chapter, and the results of all other tests are included in Appendix A.

5.3.1 Test Results

The number of faults for the 30 algorithms within the study test set and additionally for RESET configuration at TN6531 is given in Table 5.2. The total number of faults at this test is 2439. The column 'exclusive' gives the number of faults uniquely detected with the particular algorithm. The fault coverage is summarized in Fig. 5.3, where 100% refers to 2439 faults.

Table 5.2. Fault coverage of algorithms

Algorithm	|F|	FC	exclusive faults
Total	***2439***	***100%***	
SCAN	684	28,0%	0
SCAN+	727	29,8%	0
MATS	937	38,4%	0
MATS+	1047	42,9%	0
MATS++	1037	42,5%	0
March C-	1092	44,8%	0
March A	1109	45,5%	0
March B	1148	47,1%	1
Algorithm B	1908	78,2%	0
March C+	1074	44,0%	0
PMOVI	1093	44,8%	0
March 1/0	1083	44,4%	0
March TP	1110	45,5%	0
March U	1909	78,3%	1
March X	1067	43,7%	0
March Y	1056	43,3%	0
March LR	1921	78,8%	1
March LA	1414	58,0%	1
March RAW	1563	64,1%	4
March RAW1	1020	41,8%	0
March AB	1402	57,5%	0
March AB1	792	32,5%	5
March BDN	1429	58,6%	0
March SR	1898	77,8%	0
March SS	1115	45,7%	0
BLIF	1047	42,9%	0
Ham5R	783	32,1%	10
Ham5W	473	19,4%	0
March G	1232	50,5%	4
Ham_Walk	2063	84,6%	22
RESET	540	22,1%	0

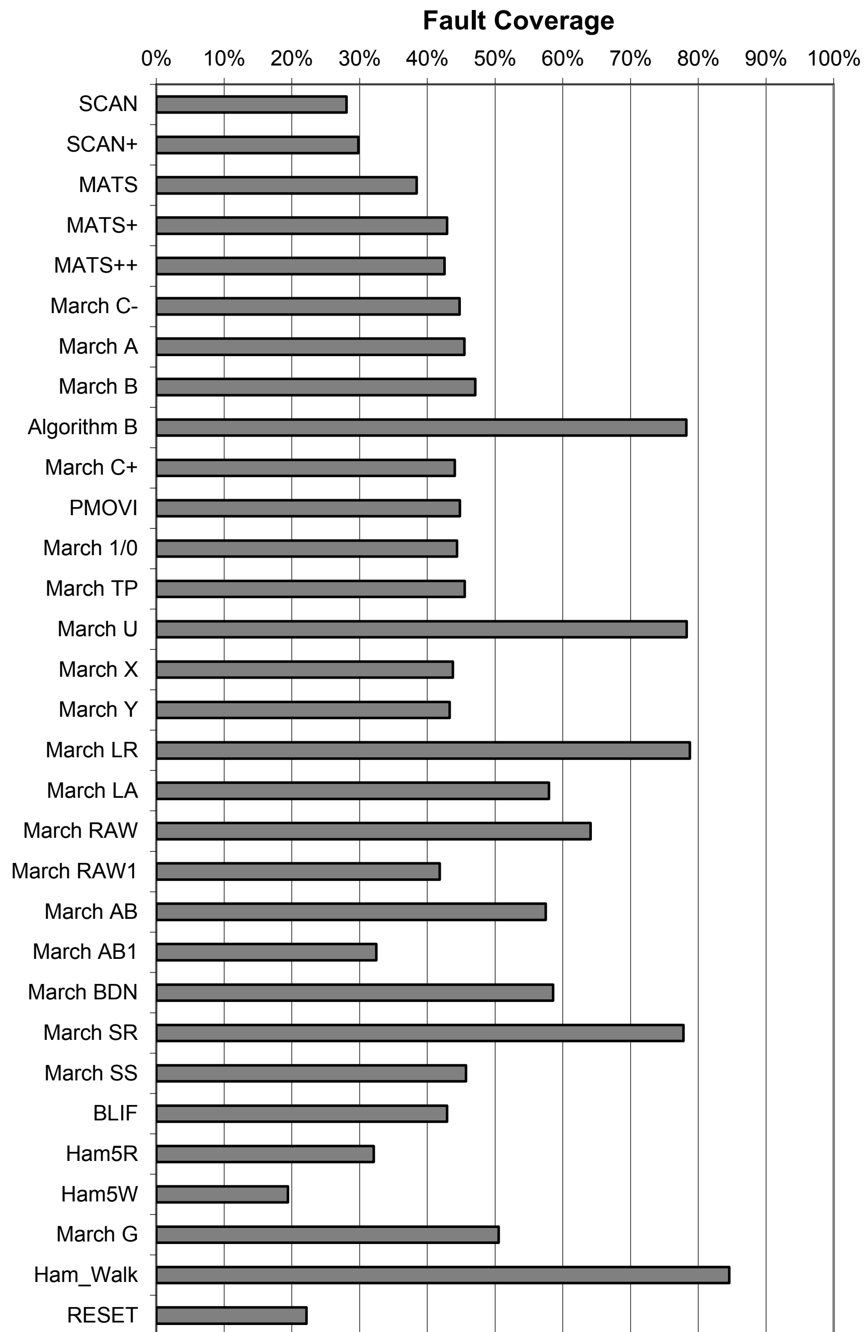

Figure 5.3. Fault coverage of algorithms

5.3.2 Evaluation of Fault Coverage

Fig. 5.3 clearly shows that none of the algorithms is able to detect all faults. The highest fault coverage is reached with Algorithm Ham_Walk at about 83%. Four other algorithms, Algorithm B, March LR, March U and March SR, are also outstanding and similar in their fault coverage with about 79%. Apparently these five algorithms seem to cover a couple of different fault models that appear frequently in the tested memories.

Traditional test algorithms (SCAN trough March Y) show much lower fault coverage in the range of 28% to 47%. Here, fewer different fault models are covered. Other recent algorithms like March RAW, March AB or March BDN, which have been developed with regard to dynamic fault models [20, 8, 28] are showing a higher fault coverage of nearly 60%. However, the relationship of algorithms and related fault models becomes not clear from this analysis. An estimation of the functional fault models that occur will be given in the following sections.

Comparing the fault coverage of the single test algorithms with the result of RESET (standard configuration of MBISTPLUS, see Chapter 4.2) shows two big issues. The fault coverage of the RESET configuration is 22%, although the same tests have already been performed during wafer test. So, the faults that are detected in this analysis slipped through wafer testing (if detected before Burn-In) or occurred due to Burn-In. The expected result of RESET at that time in the test flow should be zero. Hence, these faults are either caused by the packaging process or by differences in the environmental parameters of the test. However, the occurrence of faults with RESET shows that the RESET configuration is far from the necessary test configuration for sufficient fault coverage. Many more faults could be detected with a more efficient combination of test algorithms.

This evaluation is done for TN6531, i.e. at environmental conditions high temperature and low voltage. For all other test numbers, the results are summarized in Appendix A.

A similar distribution of fault coverage over the test algorithms can be observed for each test number. Especially dynamic fault related algorithms are showing high fault coverage independently of environmental conditions. Particularly at TN3841 (nominal voltage and low temperature), these outstanding algorithms are clearly visible as traditional algorithms are showing an extremely low fault coverage. As the voltage is at nominal value for these tests, the effect seems highly temperature dependent. For the other test series at low temperature (TN3741 at low voltage and TN3941 at high voltage), the fault coverage of traditional algorithms is much higher. However, the distribution of fault coverage of test algorithms is not significantly changed due to environmental conditions. Hence, similar test sets should be distributed for all environmental test conditions.

The analysis of the faults coverage of single test algorithms shows how effective an algorithm is independently from others. This is not yet sufficient to create an efficient set of tests, as for example two good performing tests could cover the same faults and would not improve the overall fault coverage. Therefore the effectiveness of pairs of algorithms is analyzed later in this chapter.

5.3.3 Unique Faults

Besides the total fault coverage of test algorithms, special attention should be paid to unique faults. Unique faults are exclusively detected by only one test algorithm. The coverage of unique faults is listed in Table 5.2 as exclusive faults per algorithm. The most exclusive faults are detected by Ham_Walk (22 unique faults) followed by Ham5R (10 unique faults). In these cases, obviously fault models appear that are

very special and can only be detected by special SOSs of specific test algorithms. Ham5R is the only algorithm containing five consecutive read operations in a march element. For the unique faults, this specific hammering sequence seems to be the crucial SOS that causes the cell to flip [47]. For algorithm Ham_Walk the following considerations have been done to describe reasons for the detection of unique faults. Ham_Walk is the only algorithm in the set that contains read-after-read back-to-back operations:

$$\ldots \Uparrow(r0,\ldots, r0) \ldots \quad \text{and} \quad \ldots \Uparrow(r1,\ldots, r1) \ldots \tag{7}$$

That means the last operation on one cell is read '0' (resp. read '1'), which is immediately followed again by read '0' (read '1') as first operation on the next cell. This SOS is very effective in detecting dynamic and timing related faults such as some address decoder delay faults, slow sense amplifier faults and slow pre-charge circuits. A specific explanation is not possible at that point because additionally to the SOS, algorithmic parameters are playing an important role. Especially the data background would be of special interest, but an detailed analysis of single devices would have been necessary to determine the exact circumstances.

As this is characteristic for Ham_Walk in the set of algorithms, it can be assumed that the 22 unique faults detected by Ham_Walk are related to one of these fault models. However, from the test database, no physical defects could be analyzed. For a closer evaluation and confirmation of specific fault models, the corresponding faulty devices needed to be analyzed in detail.

Also, the other unique faults seem to be related to dynamic faults. The specific SOS of Ham5R is a five times sequential read operation. This explicit hammering on one cell seems to activate and detect unique dynamic faults. The consecutive

access to one cell causes the cell to flip after a certain number of operations [47]. Each read operation decreases the charge of the memory cell until the cell flips after a certain number of sequential reads as the cell has no time to recover. The algorithms March RAW (4 unique faults) and March AB1 (5 unique faults) are also developed with scope on dynamic faults [20, 8]. The importance of dynamic faults in new SRAM technologies and the properties of March RAW, based on experimental test results, have already been shown in [48]. These algorithms are also accessing one cell sequentially without changing its value, however read and write operations are mixed. In these cases the sequentially and repeated access may cause the cell to flip as recovery time is too low.

The single unique faults of March B, March U, March LR and March LA could not be allocated to a specific SOS or specific fault model without a closer analysis of single memories.

5.4 Influence of Algorithmic Test Parameters

The use of different algorithmic test parameters has an influence on the fault coverage of each test algorithm. In Table 4.3 the combinations of algorithmic parameters are listed for all algorithms. Except for BLIF, Ham5R and Ham5W, the algorithms are used with eight different combinations of algorithmic parameters: address directions fast-x (fx) and fast-y (fy) with data backgrounds solid (so), row-stripe (rs), column-stripe (cs) and checkerboard (cb).

The fault coverage of the test algorithms, subdivided into the results for each combination of algorithmic parameters, is given in Table 5.3. The results are graphically summarized in Fig 5.4. It was not possible to split up the results for each

combination of environmental parameters, as the analysis tool did not support to distinguish between too many test parameters at once. So, the results of all test numbers are cumulated and 100% FC refers to 2712 faults.

Table 5.3. Fault coverage of algorithmic test parameters

Algorithm	Algorithmic Test Parameters							
	fx_so	fx_rs	fx_cs	fx_cb	fy_so	fy_rs	fy_cs	fy_cb
SCAN	12%	15%	21%	25%	11%	11%	24%	23%
SCAN+	14%	16%	23%	27%	13%	13%	26%	26%
MATS	16%	15%	35%	26%	14%	14%	30%	24%
MATS+	19%	15%	35%	25%	17%	15%	31%	25%
MATS++	18%	14%	35%	25%	17%	15%	33%	25%
March C-	23%	19%	39%	29%	21%	19%	36%	30%
March A	22%	19%	39%	30%	21%	19%	38%	31%
March B	19%	20%	39%	32%	22%	18%	40%	31%
Algorithm B	23%	20%	69%	32%	21%	20%	69%	32%
March C+	24%	20%	38%	30%	22%	20%	37%	31%
PMOVI	23%	20%	39%	30%	21%	20%	37%	31%
March 1/0	23%	19%	38%	29%	22%	19%	36%	30%
March TP	22%	19%	41%	30%	20%	18%	37%	31%
March U	24%	21%	69%	32%	23%	21%	69%	34%
March X	20%	16%	36%	26%	17%	17%	31%	26%
March Y	20%	16%	36%	27%	19%	17%	35%	27%
March LR	20%	21%	70%	33%	20%	21%	71%	34%
March LA	35%	29%	47%	39%	33%	28%	46%	40%
March RAW	40%	32%	50%	42%	38%	31%	50%	43%
March RAW1	20%	17%	33%	29%	18%	15%	34%	28%
March AB	34%	28%	47%	39%	32%	28%	45%	39%
March AB1	29%	21%	8%	8%	26%	21%	8%	8%
March BDN	35%	29%	47%	39%	32%	28%	47%	39%
March SR	19%	20%	68%	33%	20%	20%	69%	33%
March SS	24%	21%	39%	30%	23%	20%	38%	13%
BLIF	2%	1%	40%	4%	-	-	-	-
Ham5R	36%	-	-	-	33%	-	-	-
Ham5W	23%	-	-	-	20%	-	-	-
March G	29%	25%	45%	37%	27%	24%	44%	37%
Ham_Walk	23%	27%	71%	33%	23%	27%	69%	33%

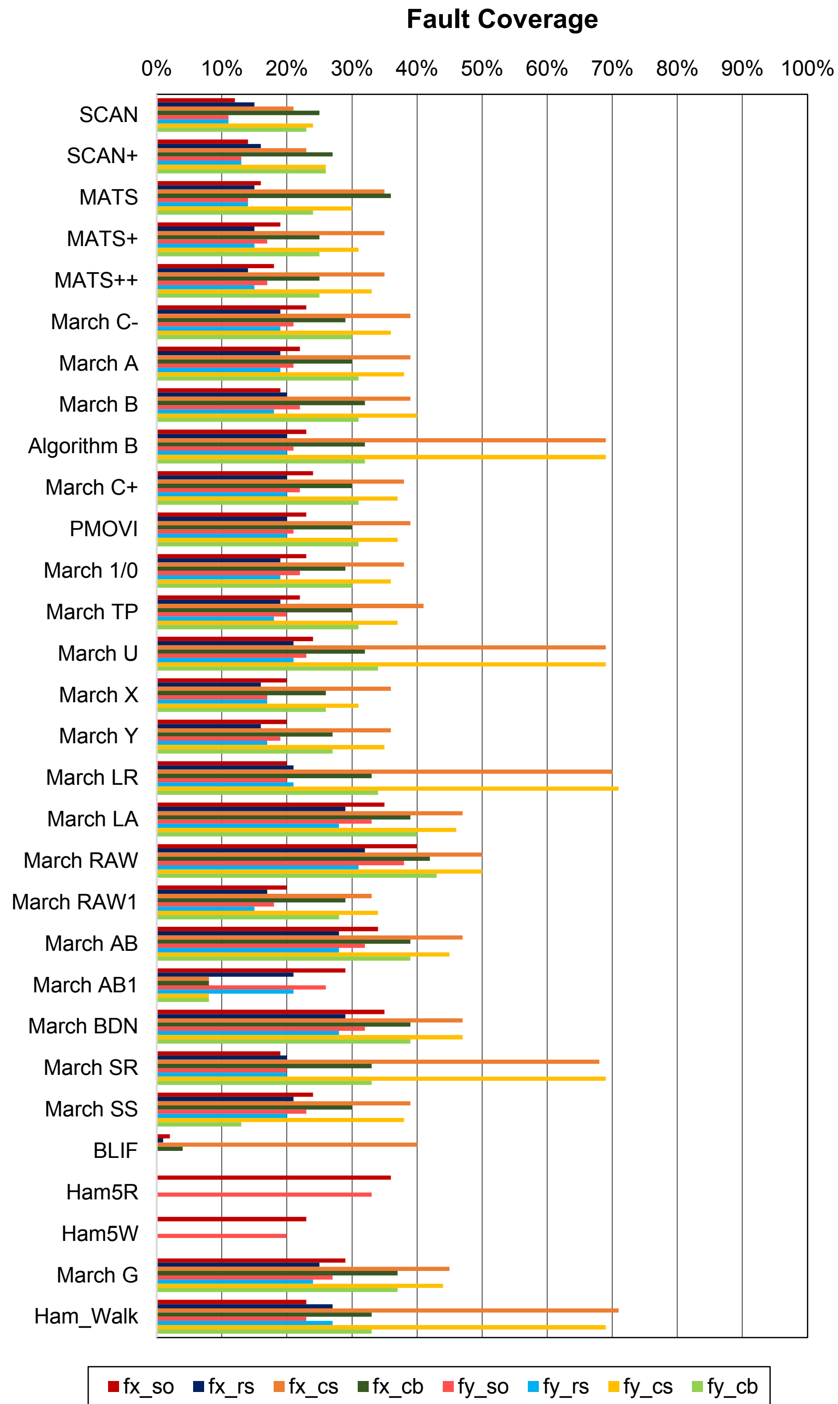

Figure 5.4. Fault coverage of algorithmic parameters

Fig. 5.4 shows the fault coverage for each combination of test algorithm and algorithmic parameters. The fault coverage of Algorithm B, March U, March LR, March SR and Ham_Walk with column-stripe data background, independently of address directions fast-x and fast-y, is most outstanding, but not with any other combination of algorithmic parameters. The fault coverage of these tests is nearly at 70%, while the other combinations of algorithmic parameters only show a fault coverage of about 20% to 30%. It is also remarkable that only these five algorithms are best with only one type of data background. Obviously column-stripe data background is most effective, but only for those algorithms. Algorithm B, March U, March LR, March SR and Ham_Walk are of similar structure and of similar fault coverage. These algorithms contain specific march sequences to detect coupling or linked faults (see Chapter 6.4.2). Apparently, column-stripe data background supports the properties of these algorithms. Due to the constant pattern along one column, and so along the bit-lines, it is likely to assume that these faults are caused by bit-line fails, e.g. crosstalk. For a closer determination, the statistical analysis is not sufficient. A more detailed analysis of single devices could determine the fault in more detail. A similar outstanding effect of one specific data background cannot be observed with any other test algorithm.

In general, the most effective data backgrounds are column-stripe and checkerboard for all algorithms, except March AB1. Here, solid and row-stripe data background are more effective; however, the overall fault coverage of March AB1 is relatively low. Unfortunately, this effect cannot be explained with the test results of this project. A more detailed analysis or simulation that also considers the memory structure and layout would have been necessary for an explanation.

The difference in the usage of fast-x and fast-y is less significant than that of different data backgrounds.

Comparing the results in Table 5.3, the fault coverage is little higher with fast-x addressing, but it cannot be said to be much more efficient.

As seen with March AB1, the effects of environmental test parameters are highly dependent on the memory structure and layout. Scrambling and mirroring may cause the differences in the results. So, the results of this analysis can only be related to those products that have been tested and cannot be generalized. For a general statement about the effectiveness of environmental test parameters, a more comprehensive analysis should be done on different products and memories of different structure and layout. However, as a result of this analysis, the most effective data backgrounds are column-stripe and checkerboard. For these data backgrounds, the background pattern alternates with each column. It is assumed that crosstalk between adjacent cells is then stronger and the different charges influence each other, so that more faults become visible with column-stripe and checkerboard data background.

5.5 Summary and Conclusions

The test algorithms have been evaluated separately, and the effectiveness of each algorithm is determined by its fault coverage. The more faults are detected by an algorithm, i.e. the higher its fault coverage, the more effective it is. Additionally, the influence of algorithmic test parameters addressing mode and data background is analyzed and effective test parameters are determined.

A few test algorithms are outstanding regarding fault coverage: Algorithm B, March U, March LR, March SR and Ham_Walk are those algorithms with highest fault coverage and so are most effective. Considering algorithmic test parameters, especially the five algorithms mentioned above, the best results are achieved with column-stripe or

checkerboard data background independently of the addressing mode. For very simple tests that only use one test algorithm (e.g. for startup tests), one of these algorithms is recommended.

Chapter 6
Efficiency of Test Algorithms

The efficiency of single test algorithms is not a sufficient base to create an efficient set of tests. The combination of algorithms needs to be analyzed separately. Two algorithms could be efficient on their own, but a combination of both algorithms does not improve the overall efficiency, if both algorithms detect the same faults.

Hence, the algorithms of the study test set are compared to each other and the efficiency and similarity of these pairs is determined. A method is described, that allows to classify similar algorithms based on their algorithmic structure and on the results of the analysis on efficiency, and a simple way is defined to find small and efficient test sets on base of statistical test results. As an outcome of this analysis and the classification of algorithms and faults, the distribution of fault models within the test results of this study is determined.

6.1 Definitions

To compare the efficiency of test algorithms, the theory of sets is used to describe the relation of number of faults, union, intersection and subsets of faults. Therefore, the set of faults detected by an algorithm is defined as F and the elements of this set are the single faults. For example, set F_U contains all faults detected by algorithm March U.

Two basic methods are used to compare the faults detected by two test algorithms: the ratio of intersection to union as degree for efficiency, and the consideration of subsets of faults as description of complete or partial coverage.

Both are used to describe the efficiency of pairs of algorithms and to classify algorithms with similar properties into sets based on statistical data analysis.

6.1.1 Union and Intersection

To define the efficiency of a pair of algorithms, it needs to be determined how many faults are detected twice and how many faults are detected newly by each algorithm. The theory of sets is used to determine union and intersection from the number of faults $|F|$, which is the cardinality of F. The union is the combined number of fault detected by two algorithms, and the intersection is the part of faults detected twice by both algorithms. If the number of faults of two algorithms "1" and "2" are $|F_1|$ and $|F_2|$, intersection (I) and union (U) are defined as (see also Fig. 6.1):

$$I = F_1 \cap F_2 \tag{8}$$

$$U = F_1 \cup F_2 \tag{9}$$

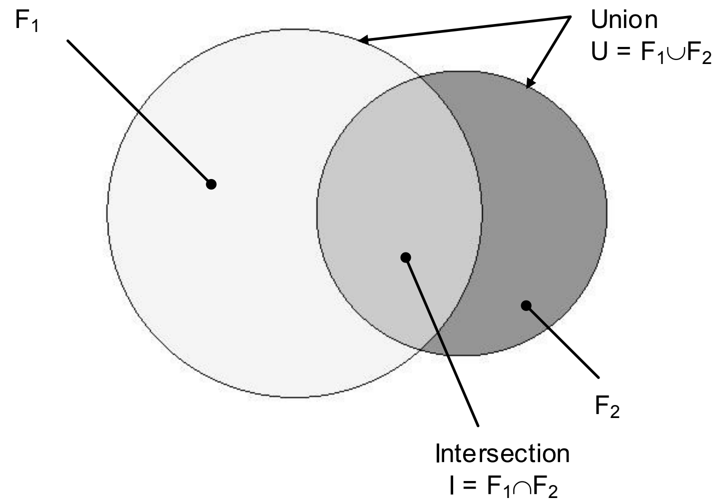

Figure 6.1. Union and intersection

Efficiency means that as many faults as possible should be detected but the number of faults detected twice should be at minimum. So, a pair of algorithms is said to be efficient if the intersection is low but the union is high at the same time. A comparison of intersection and union is needed to describe the efficiency, and so, the ratio of intersection to union is taken as a degree for efficiency. This quotient is called quotient of efficiency Q_{Eff} and is defined as:

$$Q_{Eff} = \frac{|I|}{|U|} \tag{10}$$

Q_{Eff} is then within the range of 0 to 1. Small intersection and large union means that few faults are detected twice, but the overall fault coverage is high. For this case, Q_{Eff} goes to zero. This combination of algorithms is efficient for a test set, as due to the effort of a second algorithm (additional test time) the total fault coverage increases.

If the intersection is large and approaches the union, Q_{Eff} goes to one. The combination of algorithms is inefficient

because the additional effort of a second algorithm would hardly improve the total fault coverage, but a lot of faults are detected twice. These two cases are illustrated in Fig. 6.2.

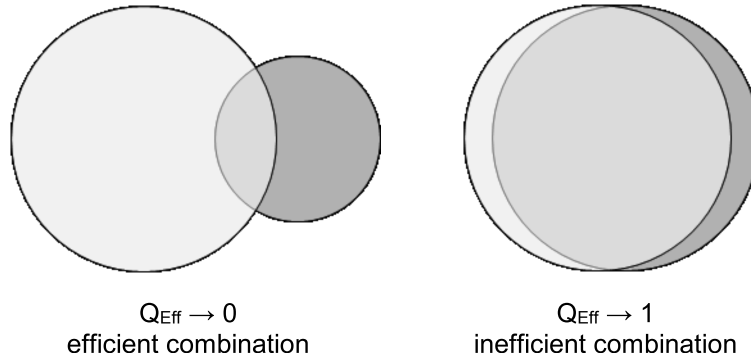

Figure 6.2. Efficiency of pairs of algorithms

Q_{Eff} as defined in (10) is only correct if F_1 and F_2 are of same cardinality. If the fault coverage is very different, the value Q_{Eff} would be falsified. To correct this, the cardinalities $|F_1|$ and $|F_2|$ are taken into account as correction factor for Q_{Eff}. Q_{Eff} is then defined as:

$$Q_{Eff} = \frac{|I|}{|U|} \cdot \frac{|F_1|}{|F_2|} \quad (11)$$

where $|F_1| > |F_2|$.

Hence, for each pair of algorithms Q_{Eff} can be determined and the efficiency of the algorithms can be stated. In the following sections, Q_{Eff} is calculated for each pair of algorithms and listed in a table. For better perceptibility, the values of Q_{Eff} are represented by different background colors. This color key is given in Fig 6.3.

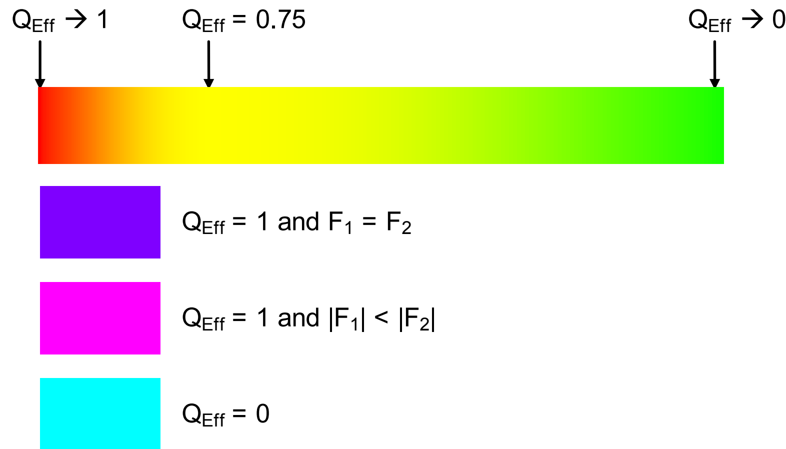

Figure 6.3. Color key

There is a continuous color scale from red ($Q_{Eff} \rightarrow 1$) to green ($Q_{Eff} \rightarrow 0$), where yellow denotes $Q_{Eff} = 0.75$, and three special cases that are illustrated in Fig. 6.4.

If no overlapping of the two sets of faults occurs, i.e. if the intersection is zero, then Q_{Eff} is zero. This case would be most efficient. In contrast, if the sets of faults are totally overlapping, Q_{Eff} is one. In this case the faults detected by one algorithm are covered by the other set of faults, while the cardinality may be both, different ($|F_1| < |F_2|$) or the sets are even the same ($F_{1\#} = F_2$). These cases would be most inefficient.

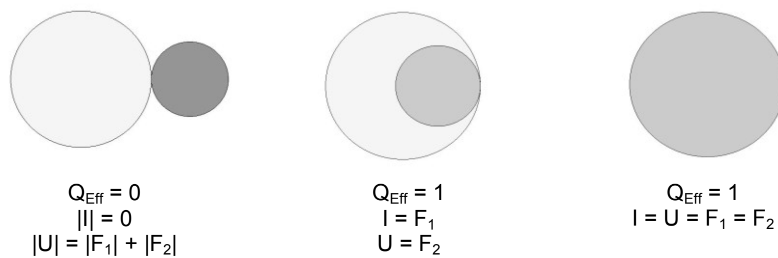

Figure 6.4. Special cases of union and intersection

The color helps to evaluate the results quickly and easily, and to get a pretty good impression which combinations are efficient and which are not.

6.1.2 Subsets and Coverage

For a more precise classification of algorithms, Q_{Eff} on its own is not sufficient enough. It needs to be considered if the fault coverage of one algorithm is a subset of another one, or if it is the same. For both cases, $Q_{Eff} \rightarrow 1$, or even $Q_{Eff} = 1$, the number of faults detected by both algorithms is compared and the following cases are distinguished.

The number of faults of the first algorithm $|F_1|$ is lower than that of the second one $|F_2|$, i.e. $|F_1| < |F_2|$. If so, F_1 is a proper subset of F_2, if $Q_{Eff} = 1$ ($F_1 \subset F_2$), and approximately a subset of F_2, if $Q_{Eff} \rightarrow 1$ ($F_1 \tilde{\subset} F_2$).

If $F_1 = F_2$ for $Q_{Eff} = 1$, and $F_1 \approx F_2$, if $Q_{Eff} \rightarrow 1$, then the sets are the same resp. approximately the same, and so are the properties of algorithms one and two. These four cases are illustrated in Fig. 6.5.

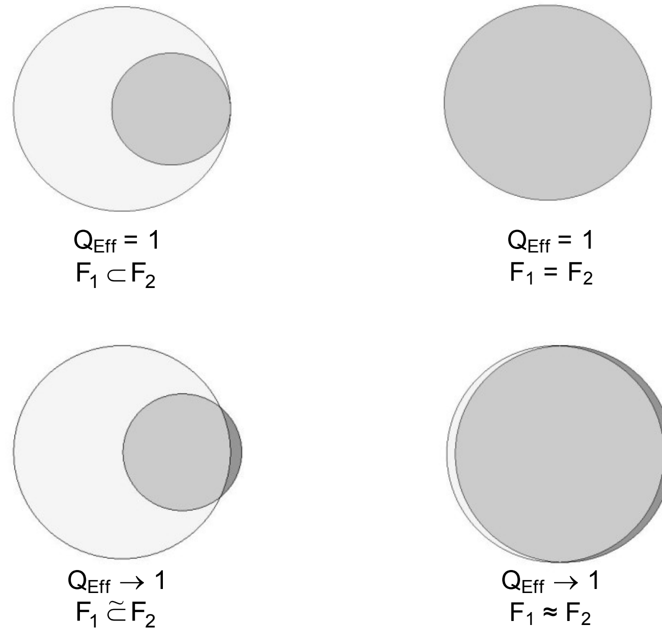

Figure 6.5. Subsets of fault coverage

Based on these definitions of efficiency and the subsets of fault coverage, the test results could be evaluated.

6.2 Evaluation Method

The analysis of Q_{Eff} and sets and subsets of faults is used to derive efficient pairs of test algorithms for memory test sets. At first, Q_{Eff} is calculated for each pair of algorithms and the results are listed. As different background colors are used to represent Q_{Eff}, a quick and easy analysis of the table can be done, and efficient pairs can easily be identified. This method is easy and allows establishing small test sets of two algorithms in a very simple way. For comprehensive test sets containing more algorithms, the analysis of only Q_{Eff} is not sufficient. To achieve the comparison of more algorithms, they are grouped into sets with similar properties, where different

functional fault models are allocated to each group. This classification and allocation is done by analyzing Q_{Eff} and subsets of fault coverage within the test results on the one hand, and by analyzing characteristic sequences of march elements and allocated fault models on the other hand. So, two independent ways are used, statistical data mining, and deterministic analysis of march elements and test algorithms. The combination of both allows to select efficient combinations of test algorithms which are able to detect as many different faults as possible but avoid redundant testing by avoiding similar test algorithms at the same time.

6.3 Efficient Pairs of Algorithms

All algorithms in this study are compared with each other pairwise and Q_{Eff} is derived to determine efficient combinations. This method is suitable for a direct comparison of two algorithms based on statistical data analysis without further knowledge about properties of test algorithms. Furthermore this way is sufficient to find a set of two efficient algorithms. This section summarizes the test results and shows how to evaluate these data.

6.3.1 Test Results

Due to the fact that at TN6531 most faults are detected and hence, the statistical analysis is most meaningful, the test results of TN6531 are presented in this section.

For each pair of algorithms the test results are taken and number of faults $|F|$, intersection $|I|$ and union $|U|$ are determined. The results are listed in Table 6.1.

Table 6.1. Effectiveness of pairs of algorithms

SCN	SCNP	MTS	MTSP	MTSPF	CM	A	B	AlgB	CP	PMOVI	1/0	TP	U	X	Y	LR	LA	RAW	RAW1	AB	AB1	BDN	SR	SS	BLIF	Ham5R	Ham5W	G	HamWk	
684	661	651	659	656	679	680	682	679	671	663	665	673	681	661	658	680	678	672	663	670	259	679	680	679	20	258	13	684	680	SCN
750	**727**	678	685	683	716	719	719	716	700	698	695	712	716	690	680	718	713	701	696	710	258	711	719	707	18	258	12	727	717	SCNP
970	986	**937**	885	890	900	906	915	926	881	885	884	906	924	879	872	916	897	888	859	889	280	904	914	888	212	273	26	928	928	MTS
1072	1089	1099	**1047**	983	992	1000	1006	1027	978	971	983	990	1032	990	972	1028	999	994	933	988	361	1004	1019	985	303	354	86	1026	1033	MTSP
1065	1081	1084	1101	**1037**	989	1002	1002	1022	976	969	982	990	1020	977	976	1016	1001	996	943	990	352	1000	1011	987	298	346	85	1022	1026	MTSPP
1097	1103	1129	1147	1140	**1092**	1032	1044	1071	1015	1015	1018	1029	1070	1000	980	1066	1044	1030	964	1034	367	1045	1059	1033	307	369	100	1077	1073	CM
1113	1117	1140	1156	1144	1169	**1109**	1069	1088	1015	1017	1017	1045	1083	1011	1000	1082	1053	1036	979	1040	365	1057	1080	1029	310	355	88	1094	1086	A
1150	1156	1170	1189	1183	1196	1188	**1148**	1116	1029	1032	1030	1052	1111	1026	1012	1112	1069	1059	990	1057	371	1078	1109	1049	335	365	93	1117	1115	B
1913	1919	1919	1928	1923	1929	1929	1940	**1908**	1065	1066	1067	1081	1834	1046	1038	1842	1187	1215	1005	1182	490	1193	1818	1087	1003	465	178	1171	1868	AlgB
1087	1101	1130	1143	1135	1151	1168	1193	1917	**1074**	1015	1020	1019	1066	987	994	1061	1040	1034	965	1036	374	1051	1060	1034	328	374	103	1057	1067	CP
1114	1122	1145	1169	1161	1170	1185	1209	1935	1152	**1093**	1020	1022	1066	981	983	1069	1039	1031	961	1030	374	1048	1066	1029	334	374	104	1065	1071	PMOVI
1102	1115	1136	1147	1138	1157	1175	1201	1924	1137	1156	**1083**	1021	1065	984	997	1065	1044	1042	958	1041	375	1048	1063	1033	331	373	104	1061	1068	1/0
1121	1125	1141	1167	1157	1173	1174	1206	1937	1165	1181	1172	**1110**	1081	996	985	1082	1049	1044	966	1038	362	1059	1075	1029	322	361	96	1086	1087	TP
1912	1920	1922	1924	1926	1931	1935	1946	1983	1917	1936	1927	1938	**1909**	1051	1037	1842	1187	1214	1008	1182	493	1203	1824	1090	1008	466	180	1169	1875	U
1090	1104	1125	1124	1127	1159	1165	1189	1929	1154	1179	1166	1181	1925	**1067**	978	1038	1004	1006	955	1010	372	1022	1033	1001	307	363	88	1044	1048	X
1082	1103	1121	1131	1117	1168	1165	1192	1926	1136	1166	1142	1181	1928	1145	**1056**	1035	1007	1011	951	1002	371	1021	1028	999	317	362	93	1034	1037	Y
1925	1930	1942	1940	1942	1947	1948	1957	1987	1934	1945	1939	1949	1988	1950	1942	**1921**	1184	1216	1006	1179	491	1201	1827	1095	1018	467	181	1168	1880	LR
1420	1428	1454	1462	1450	1462	1470	1493	2135	1448	1468	1453	1475	2136	1477	1463	2151	**1414**	1344	983	1334	626	1355	1189	1069	417	599	325	1103	1246	LA
1575	1589	1612	1616	1604	1625	1636	1652	2256	1603	1625	1604	1629	2258	1624	1608	2268	1633	**1563**	979	1331	743	1350	1215	1072	462	731	449	1097	1279	RAW
1041	1051	1098	1134	1114	1148	1150	1178	1923	1129	1152	1145	1164	1921	1132	1125	1935	1451	1604	**1020**	987	361	997	1003	981	283	350	82	1007	1012	RAW1
1416	1419	1450	1461	1449	1460	1471	1493	2128	1440	1465	1444	1474	2129	1459	1456	2144	1482	1634	1435	**1402**	628	1348	1183	1062	424	595	317	1103	1246	AB
1217	1261	1449	1478	1477	1517	1536	1569	2210	1492	1511	1500	1540	2208	1487	1477	2222	1580	1612	1451	1566	**792**	632	498	386	240	639	407	388	546	AB1
1434	1445	1462	1472	1466	1476	1481	1499	2144	1452	1474	1464	1480	2135	1474	1464	2149	1488	1642	1452	1483	1589	**1429**	1199	1077	427	602	325	1110	1261	BDN
1902	1906	1921	1926	1924	1931	1927	1937	1988	1912	1925	1918	1933	1983	1932	1926	1992	2123	2246	1915	2117	2192	2128	**1898**	1085	999	469	184	1167	1851	SR
1120	1135	1164	1177	1165	1174	1195	1214	1936	1155	1179	1165	1196	1934	1181	1172	1941	1460	1606	1154	1455	1521	1467	1928	**1115**	349	389	115	1081	1096	SS
1711	1756	1772	1791	1786	1832	1846	1860	1952	1793	1806	1799	1835	1948	1807	1786	1950	2044	2148	1784	2025	1599	2049	1946	1813	**1047**	218	179	374	1027	BLIF
1209	1252	1447	1476	1474	1506	1537	1566	2226	1483	1502	1493	1532	2226	1487	1477	2237	1598	1615	1453	1590	936	1610	2212	1509	1612	**783**	458	383	500	Ham5R
1144	1188	1384	1434	1425	1465	1494	1528	2203	1444	1462	1452	1487	2202	1452	1436	2213	1562	1587	1411	1558	858	1577	2187	1473	1341	798	**473**	108	212	Ham5W
1232	1232	1241	1253	1247	1247	1247	1263	1969	1249	1260	1254	1256	1972	1255	1254	1985	1543	1698	1245	1531	1636	1551	1963	1266	1905	1632	1597	**1232**	1187	G
2067	2073	2072	2077	2074	2082	2086	2096	2103	2070	2085	2078	2086	2097	2082	2082	2104	2231	2347	2071	2219	2309	2231	2110	2082	2083	2346	2324	2108	**2063**	HamWk

The algorithms of the study test set are listed horizontally and vertically. The main diagonal contains the number of faults covered by each algorithm (see also Table 5.2). For each pair, the intersection is entered above the main diagonal and the union below.

Q_{Eff} is calculated for each pair of algorithms according to formula (11) and is represented as background color according to the key in Fig 6.3. An uncolored table containing the values of Q_{Eff}, as well as the results for test numbers during the study are provided in Appendix A.2.

6.3.2 Data Evaluation

With help of Table 6.1, a quick and easy overview is possible to estimate the effectiveness of pairs of algorithms. One algorithm can easily be compared to each other by evaluating the results of one column or row. Each entry in the table that is marked red or orange denotes a inefficient pair of algorithms, where yellow and green entries are related to efficient combinations of algorithms.

Examples of an efficient and an inefficient combination based on March U are given in Fig. 6.6. March U is compared to Ham5R on the one hand and to March SR on the other hand.

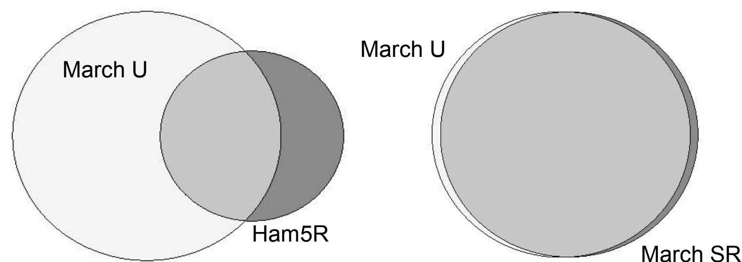

Figure 6.6. Efficient and inefficient algorithms

For March U combined with Ham5R, the following values are determined from table 6.1:

- Fault coverage of March U: $|F_U| = 1909$
- Fault coverage of Ham5R: $|F_{5R}| = 783$
- Intersection: $|I| = 466$
- Union: $|U| = 2226$

Q_{Eff} for the combination of March U and Ham5R then is:

$$Q_{Eff}(U,5R) = \frac{|I|}{|U|} \cdot \frac{|F_{5R}|}{|F_U|} = \frac{466}{2226} \cdot \frac{1909}{783} = 0.51 \qquad (12)$$

For the combination of March U and March SR, the following values are determined:

- Fault coverage of March U: $|F_U| = 1909$
- Fault coverage of March SR: $|F_{SR}| = 1898$
- Intersection: $|I| = 1824$
- Union: $|U| = 1934$

Q_{Eff} for the combination March U with March SR is:

$$Q_{Eff}(U,SR) = \frac{|I|}{|U|} \cdot \frac{|F_{SR}|}{|F_U|} = \frac{1824}{1934} \cdot \frac{1909}{1898} = 0.93 \qquad (13)$$

The values of Q_{Eff} are represented in Fig. 5.3 as green background color for the combination of March U and Ham5R, and as orange background for March U and March SR.

Likewise, Q_{Eff} is derived for each pair of algorithms, and interpreted as color in Table 6.1. But more than the single values, the table gives an overview of the efficiency of the algorithms. The algorithms can be characterized by analyzing Table 6.1 column by column. Each column represents one algorithm. If the entries in one column are predominantly orange or red (e.g. Algorithm B or March U), these algorithms

are covering most of the other algorithms or are covered by other algorithms (e.g. SCAN or MATS). In those cases, the values of union and intersection have to be considered. So, if algorithms are covering others, they can be assumed to be more effective than other algorithms. On the other hand, if a column is predominantly green, the algorithm detects additional other faults than the remaining algorithms. These algorithms seem to be March AB1, BLIF, Ham5R and Ham5W.

However, these algorithms are also covered by other ones. E.g. the column of March AB1 is predominantly green, but the faults detected by March AB1 are nearly completely covered by March RAW, which covers 94% of the faults of March AB1 (743 faults of 792). Anyway, this knowledge about coverage is very helpful for the estimation of faults. A special case occurs with March G and SCAN resp. SCAN+. Here, March G completely covers the other algorithms. A combination of these algorithms in a test set would be useless, as neither SCAN nor SCAN+ would improve the total fault coverage of the test set.

Deriving Q_{Eff} and analyzing the results of Table 6.1 does not yet consider the coverage of specific fault models. Only the pure ratio of union, intersection and fault coverage is taken into account to determine efficient combinations of algorithms and to provide a base for the further analysis and classification of test algorithms and functional faults. However, for simple test sets containing two test algorithms, the method by comparing Q_{Eff} for each pair of algorithms in the study may be sufficient if irredundant testing is desired, and a quick method should be used to define a simple test set.

6.4 Classification of Algorithms

The analysis of pairs of algorithms has shown that there are pairs of algorithms where the set of faults detected by one algorithm covers that of another algorithm completely or

partially. Using such a pair of algorithms together in one test set means redundant testing as one algorithm may replace the other one, and so unnecessary additional test time would be needed. This is inefficient in productive memory testing. Hence, test algorithms with similar fault coverage or similar properties should be avoided. Therefore the algorithms of this study are analyzed on their structure and fault coverage and are classified into groups. Each group of algorithms is then allocated to a specific group of functional fault models.

The classification is done in two independent ways. Algorithms of similar structure containing characteristic march elements can be grouped and allocated to corresponding sets of functional faults. The information about characteristic march elements and corresponding functional fault is taken from literature where march algorithms are described in detail. On the other hand, the statistical analysis of test data is taken to group algorithms based on similar fault coverage and the coverage of subsets of faults.

6.4.1 Similar Fault Coverage and Subsets

For the analysis of similar fault coverage and subsets of faults, the results of the statistical analysis of productive test data are taken into account. The structure or structural properties of march elements are not of interest, but only the experimental test results are considered. As the test results of TN6531 are most meaningful, the results of Table 6.1 are taken for this analysis.

Algorithm B, March U, March LR, March SR and Ham_Walk form a set of similar algorithms. For each pair of algorithms in this set Q_{Eff} is derived and entered in Table 6.2 together with $|F|$ of each algorithm on the main diagonal.

Table 6.2. Fault coverage and Q_{Eff}

	Alg. B	March U	March LR	March SR	Ham_Walk
Algorithm B	**1908**	0.925	0.933	0.919	0.960
March U		**1909**	0.932	0.925	0.966
March LR			**1921**	0.928	0.960
March SR				**1898**	0.954
Ham_Walk					**2063**

The number of faults detected by these algorithms is approximately the same, except for Ham_Walk, where $|F_{Ham_Walk}|$ is bigger than that of the other algorithms. However, $Q_{Eff} \rightarrow 1$ for each pair and is approximately the same. This means that Ham_Walk covers the other algorithms in this set:

$$|F_{AlgB}| \approx |F_U| \approx |F_{LR}| \approx |F_{SR}| \qquad (14)$$

and

$$\{F_{AlgB}, F_U, F_{LR}, F_{SR}\} \approxeq\subset F_{Ham_Walk} \qquad (15)$$

These relationships are summarized in Fig 6.7. The Venn diagrams show the proportion of union and intersection related to March LR. Any other combination of these algorithms results in a similar Venn diagram.

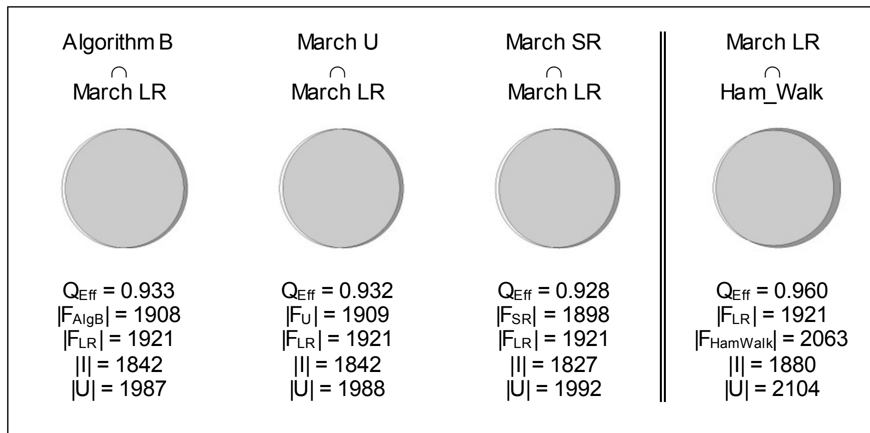

Figure 6.7. Venn diagrams I

The comparison of March LR and Ham_Walk is of special interest. As described above, the structure of these algorithms is almost the same. The main difference is given by the RAR_{wa} sequence of Ham_Walk. It should be expected, that March LR is completely covered by Ham_Walk. However, this is not the case. A couple of 41 faults are still detected by March LR but not by Ham_Walk. The reason for that cannot be explained on base of statistical data analysis. These faults needed to be analyzed explicitly to exclude variation of operation conditions (e.g. minimal temperature variance) during the tests. On the other hand, the faults that are additionally detected by Ham_Walk can almost certainly be assumed to be detected by the RAR_{wa} sequence. However, any influence by the test process itself can also not be excluded.

SCAN and SCAN+ are the only algorithms in this analysis, where the faults are completely covered by March G. The set of faults detected by SCAN and SCAN+ are even a proper subset of the faults detected by March G (see Fig. 6.8).

$$F_{SCAN} \subset F_G \quad \text{and} \quad F_{SCAN+} \subset F_G \tag{16}$$

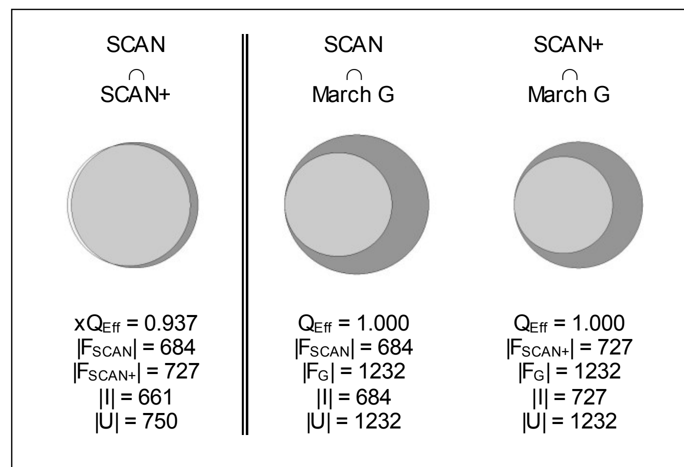

Figure 6.8. Venn diagrams II

Although the structure of SCAN is a proper subset of SCAN+ (see Table 3.2), F_{SCAN} is not a proper subset of F_{SCAN+}. So there are faults that are detected by SCAN but not by SCAN+. This is remarkable, as it should be expected that, from their structure, SCAN+ completely covers SCAN. However, it cannot be explained, as the statistical analysis is not sufficient to explain the behavior of single faults. A more comprehensive analysis of such faulty devices would be necessary to identify the fault model behind this effect, and to analyze if such a faulty behavior is repeatable or if it depends on environmental disturb factors (e.g. variations of environmental test parameters).

The following results apply for hammering algorithm Ham5R and Ham5W. Both, Ham5R and Ham5W detect approximately a subset of those faults detected by March RAW (see Fig. 6.9).

$$F_{Ham5W} \tilde{\subset} F_{Ham5R} \quad (17)$$

$$\{F_{Ham5W}, F_{Ham5R}\} \tilde{\subset} F_{RAW} \quad (18)$$

Ham5R and March RAW are also compared to Ham_Walk and it shows up, that there are many faults that are not detected by Ham_Walk. Obviously there are fault models that cannot be detected by Ham_Walk but well by March RAW and Ham5R (Fig. 6.9).

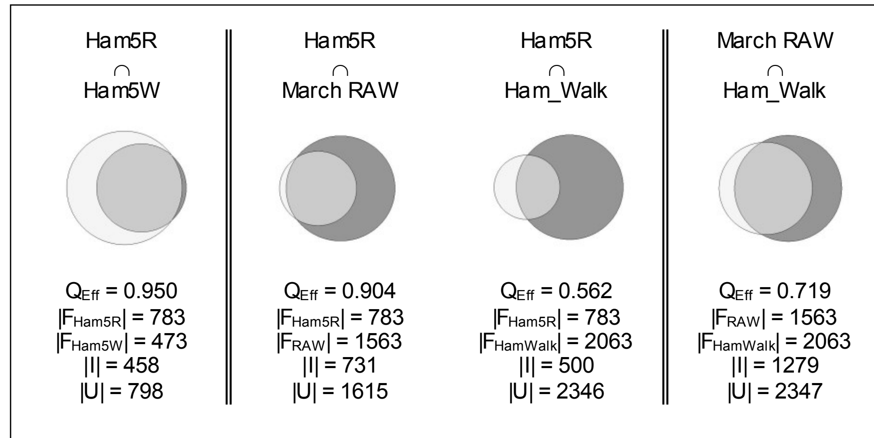

Figure 6.9. Venn diagrams III

A few algorithms are developed especially for dynamic faults. In the study test set, these are March RAW, March RAW1, March AB, March AB1 and March BDN. The statistical analysis shows that March RAW, March AB and March BDN are very similar (Fig. 6.10). From [8] and [28] this could be expected as March AB and March BDN are developed to detect the same set of dynamic faults as March RAW. The productive test results confirm this assumption. So is:

$$F_{AB} \approx F_{BDN} \approx F_{RAW} \qquad (19)$$

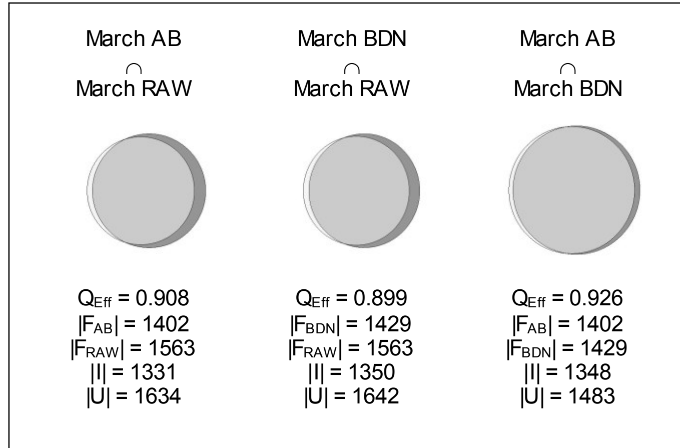

Figure 6.10. Venn diagrams IV

Furthermore is:

$$F_{RAW1} \tilde{\subset} F_{RAW} \quad (20)$$

and

$$F_{AB1} \tilde{\subset} F_{AB} \quad (21)$$

The fault coverage of March RAW1 is a subset of March RAW and the March AB1 is a subset of March AB1 (see Fig. 6.11). It is known from [7] and [8] respectively, that March RAW1 and March AB1 are related to single-cell dynamic faults, whereas March RAW and March AB are also related to two-cell dynamic faults. Hence, the set of faults allocated to March RAW1 and March AB1 is a subset of the set of faults allocated to March RAW and March AB. The assumption that March RAW1 and March AB1 are covered by March RAW and March AB can so be shown by the experimental results (Fig. 6.11).

However, the assumption that March AB1 reaches the same results as March RAW1 [8] cannot be confirmed. The Venn diagram in Fig. 6.11 shows that the two algorithms are very different in their fault coverage, and Q_{Eff} = 0.320 also shows low similarity. But March AB1 is nevertheless covered by March RAW. So the set of faults detected by March AB1 is

still a subset of dynamic faults, but March AB1 cannot be used to replace March RAW1.

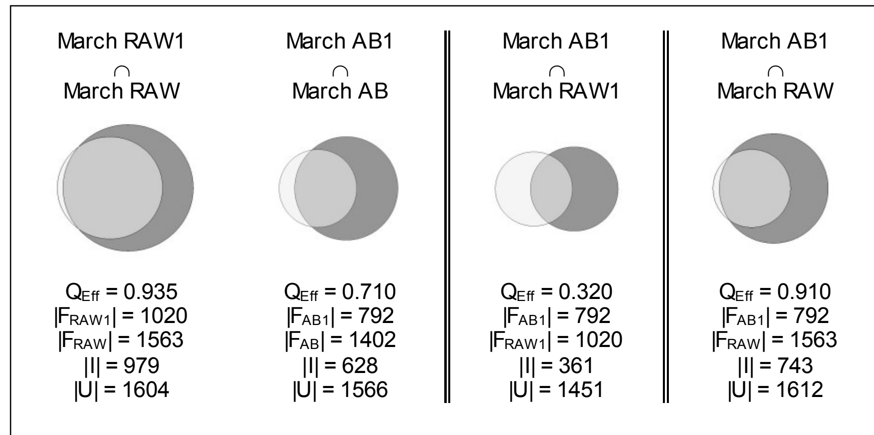

Figure 6.11. Venn diagrams V

Another very interesting comparison is the intersection of SCAN and Ham5W. With Q_{Eff} = 0.016, this is obviously one of the most efficient combinations within the test results of this study. The sets of faults detected by these algorithms are nearly not overlapping. So, the performance is completely different, which is important to know as a criterion for exclusion for the classification of algorithms and allocation of functional fault models. For an overall test performance, SCAN and Ham5W seem not to play an important role, as both are almost covered by March RAW. And March RAW even covers the combination of both algorithms (see Fig. 6.12).

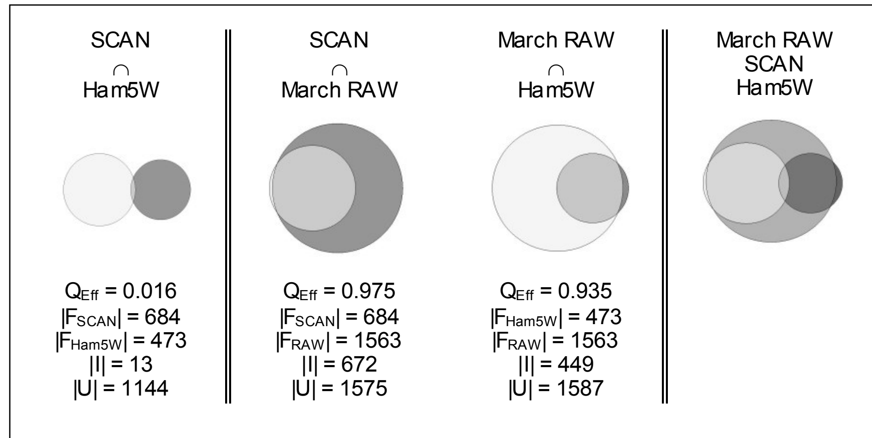

Figure 6.12. Venn diagrams VI

The biggest part of algorithms has a fault coverage which is approximately a subset of that of Algorithm B, March U, March LR, March SR or Ham_Walk. The algorithms which are covered by this set are: SCAN, SCAN+, MATS, MATS+, MATS++, March A, March B, March C-, March C+, PMOVI, March 1/0, March TP, March X, March Y, March SS and BLIF. Furthermore, $|F_{SCAN}|$ and $|F_{SCAN+}|$ are relatively low and is approximately covered by the other algorithms in this list.

Some algorithms show up to be important in classification as they stand out in their performance. These algorithms are March RAW, Ham_Walk, Ham5R and March G. For test set development, these algorithms are of special interest and seem to be indispensable.

6.4.2 Characteristic March Elements

In [4, 19], conditions are described that characterize specific sequences of march elements that are allocated to specific fault models. Some of these conditions are used to identify similar test algorithms and to classify them. The indications of these conditions "5", "5S", "6", "6S" and "6SD" are taken from [4] and [19].

Condition 5S is to detect all simple coupling faults [4] in SRAM memories and requires one of the following sequences of march elements:

$$\Uparrow(rx,\ldots,w\bar{x},\ldots,wx);\Updownarrow(rx,\ldots)$$
$$\text{and} \tag{22}$$
$$\Uparrow(r\bar{x},\ldots,wx,\ldots,w\bar{x});\Updownarrow(r\bar{x},\ldots)$$

or

$$\Downarrow(rx,\ldots,w\bar{x},\ldots,wx);\Updownarrow(rx,\ldots)$$
$$\text{and} \tag{23}$$
$$\Downarrow(r\bar{x},\ldots,wx,\ldots,w\bar{x});\Updownarrow(r\bar{x},\ldots)$$

where $x \in \{0,1\}$.

This sequence is performed exactly only by March LR. So, according to this condition, only March LR is able to detect all simple coupling faults. However, a couple of algorithms show the sequences in a slightly modified form. For example there is an additional read operation at the end of the first march element, the up/down-direction is different, or the second part of the sequence is missing. These algorithms are at least similar to March LR concerning this characteristic march sequence, but would not detect all simple coupling faults. In Table 6.3 algorithms are listed, and the characteristic march elements are marked. It is also listed which algorithm requires the condition totally by "5S", if up/down-direction are inverted by "(5S)", or if the read-write-sequence is incomplete by "((5S))".

Table 6.3. March algorithms for simple coupling faults

Algorithm	Sequence	Cond.
March LR	{⇳(w0);⇩(r0,w1);⇧**(r1,w0**,r0,**w1);**⇧**(r1**,w0);⇧**(r0,w1**,r1,**w0);**⇧**(r0)**}	5S
March A	{⇳(w0);⇧(r0,w1,w0,w1);⇧**(r1,w0,w1);**⇩**(r1**,w0,w1,w0);⇩**(r0,w1,w0)**}	((5S))
March B	{⇳(w0);⇧(r0,w1,r1,w0,r0,w1);⇧**(r1,w0,w1):**⇩**(r1**,w0,w1,w0);⇩**(r0,w1,w0)**}	((5S))
Algorithm B	{⇳(w0);⇧(r0,w1,w0,w1);⇧**(r1,w0**,r0,**w1**);⇩**(r1**,w0,w1,w0);⇩**(r0,w1**,r1,**w0**)}	((5S))
March U	{⇳(w0);⇧**(r0,w1**,r1,**w0**);⇧**(r0**,w1);⇩(r1,w0,);⇩**(r0,w1**,r1,**w0**);⇩**(r0)**}	(5S)
March SR	{⇩(w0);⇧**(r0,w1**,r1,**w0**);⇧**(r0**,r0);⇧(w1);⇩**(r1,w0**,r0,**w1**);⇩**(r1**,r1)}	(5S)
March G	{⇳(w0);⇧(r0,w1,r1,w0,r0,w1);⇧**(r1,w0,w1);**⇩**(r1**,w0,w1,w0);⇩**(r0,w1,w0)**; D;⇧**(r0**,w1,r1);D;⇧(r1,w0,r0)}	((5S))
Ham_Walk	{⇧(w1);⇧(w0);⇧**(r0,w1**,r1,**w0**,r0);⇧**(r0**,w1);⇧**(r1,w0**,r0,**w1**,r1);⇧**(r1)**}	5S

For March A, March B and Algorithm B, the second part of the sequence is incomplete and the up/down-direction is changed. For Algorithm B, March U, March SR and March G, the sequences are complete but the up/down-direction is changed, and for March G, the second part of march elements is interrupted by a delay time *D*.

Ham_Walk meets the requirement that both parts of the sequence are performed in up-direction. However, there is an additional read operation at the end of march elements M2 and M4.

Conditions for detecting linked faults are described in [19]. The space of linked faults is complex and factors like address order and sets of simple coupling faults which are linked to each other are playing an important role in detecting linked faults, three conditions are defined. Condition 6, conditions 6S and condition 6SD are detecting linked faults, where condition 6SD is the most comprehensive one and covers conditions 6S and 6, where condition 6S already covers condition 6. The following sequences of march elements are defined for condition 6SD.

$$\Uparrow\left(rx, w\bar{x}, r\bar{x}, \ldots, wx\right); \Uparrow\left(rx, \ldots\right) \quad \text{and} \quad \Uparrow\left(r\bar{x}, wx, rx, \ldots, w\bar{x}\right); \Uparrow\left(r\bar{x}, \ldots\right) \tag{24}$$

or

$$\Downarrow(rx, w\bar{x}, r\bar{x}, \ldots, wx); \Downarrow(rx, \ldots) \quad \text{and} \quad \Downarrow(r\bar{x}, wx, rx, \ldots, w\bar{x}); \Downarrow(r\bar{x}, \ldots) \tag{25}$$

Condition 6S is developed from condition 6 by adding a second march element to the sequences, and condition 6SD is developed from condition 6S by adding a read operation. The structure of conditions 6 and 6S are given below.

Conditions 6:

$$\Uparrow(rx, w\bar{x}, \ldots, wx) \text{ and } \Uparrow(r\bar{x}, wx, \ldots, w\bar{x}) \quad \text{or} \quad \Downarrow(rx, w\bar{x}, \ldots, wx) \text{ and } \Downarrow(r\bar{x}, wx, \ldots, w\bar{x}) \tag{26}$$

Conditions 6S:

$$\Uparrow(rx, w\bar{x}, \ldots, wx); \Uparrow(rx) \text{ and } \Uparrow(r\bar{x}, wx, \ldots, w\bar{x}); \Uparrow(r\bar{x}) \quad \text{or} \quad \Downarrow(rx, w\bar{x}, \ldots, wx); \Downarrow(rx) \text{ and } \Downarrow(r\bar{x}, wx, \ldots, w\bar{x}); \Downarrow(r\bar{x}) \tag{27}$$

An important characteristic of these conditions is that all march elements in one sequence are performed either in up or down direction. The direction must not change; otherwise some linked fault may be masked and undetected.

Condition 6SD is exactly used by March LR. There are also a couple of other algorithms which are very similar to these conditions, but do not exactly meet the requirements, i.e. the up/down-direction changes during a characteristic sequence. So these algorithms would detect some linked faults but with exceptions. In Table 6.4 the algorithms are listed and the characteristic sequences are highlighted. If the up-down-direction of a condition is incorrect, the condition is given in brackets.

Table 6.4. March algorithms for linked faults

Algorithm	Sequence	Cond.
March LR	{⇕(w0);⇩(r0,w1);**⇧(r1,w0,r0,w1);⇧(r1**,w0);**⇧(r0,w1,r1,w0);⇧(r0)**}	6SD
March A	{⇕(w0);⇧(r0,w1,w0,w1);**⇧(r1,w0,w1);⇩(r1**,w0,w1,w0);**⇩(r0,w1,w0)**}	(6)
March B	{⇕(w0);⇧(r0,w1,r1,w0,r0,w1);**⇧(r1,w0,w1):⇩(r1**,w0,w1,w0);**⇩(r0,w1,w0)**}	(6)
Algorithm B	{⇕(w0);⇧(r0,w1,w0,w1);**⇧(r1,w0,r0,w1);⇩(r1**,w0,w1,w0);**⇩(r0,w1,r1,w0**)}	(6S)
March U	{⇕(w0);**⇧(r0,w1,r1,w0**);**⇧(r0**,w1);⇩(r1,w0,);**⇩(r0,w1,r1,w0**);**⇩(r0)**}	(6SD)
March SR	{⇩(w0);**⇧(r0,w1,r1,w0**);**⇧(r0**,r0);⇧(w1);**⇩(r1,w0,r0,w1**);**⇩(r1**,r1)}	(6SD)
March G	{⇕(w0);⇧(r0,w1,r1,w0,r0,w1);**⇧(r1,w0,w1);⇩(r1**,w0,w1,w0);**⇩(r0,w1,w0)**; D;**⇧(r0**,w1,r1);D;⇧(r1,w0,r0)}	(6)
Ham_Walk	{⇧(w1);⇧(w0);**⇧(r0,w1,r1,w0**,r0);**⇧(r0**,w1);**⇧(r1,w0,r0,w1**,r1);**⇧(r1)**}	6SD

March A, March B and Algorithm B nearly meet the requirements of condition 6, however the addressing direction changes during the sequence. The same occurs at March U, March SR and March G, where the sequences nearly meets the requirements of condition 6S. March G has also an additional delay time within the sequence. Ham_Walk is closest to condition 6SD, but there is an additional read operation within the sequence.

Another class of algorithms is related to dynamic faults. These algorithms are previously described in literature [20, 27, 28]. A characteristic sequence performed by those algorithms is:

$$\begin{gathered} \Updownarrow(\ldots, wx, rx); \Updownarrow(rx, \ldots) \\ \text{and} \\ \Updownarrow(\ldots, w\overline{x}, r\overline{x}); \Updownarrow(r\overline{x}, \ldots) \end{gathered} \tag{28}$$

Additionally to these sequences, delay time between march elements enables detecting faults, and also traditional test algorithms are able to detect a reduced set of dynamic faults [7]. All algorithms that can be allocated to detect dynamic faults are listed in Table 6.5.

Table 6.5. March algorithms for dynamic faults

Algorithm	Sequence
March RAW	{⇕(w0);⇑(r0,w0,r0,r0,w1,r1);⇑(r1,w1,r1,r1,w0,r0);⇓(r0,w0,r0,r0,w1,r1) ⇓(r1,w1,r1,r1,w0,r0)⇕(r0)}
March RAW1	{⇕(w0);⇕(w0,r0);⇕(r0);⇕(w1,r1);⇕(r1);⇕(w1);⇕(w1,r1);⇕(w0);⇕(w0,r0)}
March AB	{⇕(w1);⇓(r1,w0,r0,w0,r0);⇓(r0,w1,r1,w1,r1);⇑(r1,w0,r0,w0,r0); ⇑(r0,w1,r1,w1,r1);⇕(r1)}
March AB1	{⇕(w0);⇕(w1,r1,w1,r1,r1);⇕(w0,r0,w0,r0,r0)}
March BDN	{⇑(w0);⇓(r0,w1,r1.w1,r1);⇓(r1,w0,r0,w0,r0);⇑(r0,w1,r1,w1,r1); ⇑(r1,w0,r0,w0,r0);⇑(r0)}
March LA	{⇕(w0);⇑(r0,w1,w0,w1,r1);⇑(r1,w0,w1,w0,r0);⇓(r0,w1,w0,w1,r1); ⇓(r1,w0,w1,w0,r0)⇓(r0)}
Ham_Walk	{⇑(w1);⇑(w0);⇑(r0,w1,r1,w0,r0);⇑(r0,w1);⇑(r1,w0,r0,w1,r1);⇑(r1)}
March G	{⇕(w0);⇑(r0,w1,r1,w0,r0,w1);⇑(r1,w0,w1);⇓(r1,w0,w1,w0);⇓(r0,w1,w0); D;⇑(r0,w1,r1);D;⇑(r1,w0,r0)}

The most important algorithms for detecting dynamic faults in this list are March RAW, March AB, March AB1, March BDN and March G. These algorithms have been developed to predominantly detect dynamic faults. The characteristic march sequence is also included in algorithms March LA and Ham_Walk. The set of dynamic faults is not restricted to the algorithms in this list. Other algorithms like March C-, March B, PMOVI, March U, March SR or March LR partially detect dynamic faults as well [7]. Ham_Walk is the only algorithm which contains a characteristic, walking read-after-read (RaR_{wa}) operation in march elements M2 and M4. Walking means that after applying the march sequence to the cell, it is in the same state as before.

$$\Updownarrow (rx, \ldots, rx) \tag{29}$$

Apart from this and an additional initializing write operation, Ham_Walk is the same as March LR. This RAR_{wa} operation is very effective to detect dynamic faults and timing related faults like some address decoder delay faults, slow sense amplifier and slow pre-charge circuit faults. Hence, Ham_Walk should be included to the set of algorithms detecting these dynamic faults.

Besides the pure analysis of characteristic march elements, algorithms are already described in literature and related to specific FFMs [1]. The allocation is summarized in Table 6.6.

Table 6.6. Fault model coverage of traditional march tests

Algorithm	Fault Models
MATS	some AF, SAF
MATS+	AF, SAF
MATS+	AF, SAF, TF
MATS++	AF, SAF, TF
March X	AF, SAF, TF, CFin
March C-	AF, SAF, TF, CFin, CFid
March A	AF, SAF, TF, CFin, linked CFid
March Y	AF, SAF, CFin, TF linked with CFin
March B	AF, SAF, CFin, linked CFid, TF linked with CFin

6.4.3 Grouping and Classification

The grouping and classification of march test algorithms is based on similar characteristics on one hand – derived from literature research, and on statistical determination on the other hand – derived from the statistical analysis. At the same time, the groups of algorithms are allocated to sets of functional fault models, which are predominantly detected by those algorithms. For the analysis in this section, the space of functional faults is limited to the following four supersets:

- simple, static single cell faults (SSs),
- simple, static coupling faults (CFs),
- linked faults (LFs), and
- dynamic faults (DFs).

Five sets of algorithms could be determined and allocated to functional fault models. These sets are listed in Table 6.7.

Table 6.7. Sets of algorithms and functional faults

	Set of Algorithms	Set of FFMs
Set I	SCAN, SCAN+	simple static single-cell faults
Set II	MATS, MATS+, MATS++, March X, March Y, PMOVI, March 1/0, March C-, March C+, March TP, March A, March B, March SS, BLIF	simple static single-cell faults some simple CFs
Set III	Algorithm B, March U, March SR	simple static single-cell faults most simple CFs some linked faults
Set IV	March LA, March LR, Ham_Walk	simple static single-cell faults all simple CFs linked faults some dynamic faults
Set V	March RAW, March RAW1, March AB, March AB1, March BDN, Ham5R, Ham5W, March G	simple static single-cell faults simple static coupling some linked faults dynamic faults

Each of the sets is allocated to a certain number of functional fault models which are predominantly detected, but not limited to those functional faults. E.g., as described in [7], algorithms of sets II, III and IV also detect dynamic faults, however only partially.

This classification of algorithms and fault models is base for efficient selection of test sets, as similar algorithms can be avoided, and so redundant testing. From the above findings about similar characteristics and fault coverage of test algorithms, the following conclusions can be drawn:

- Set I and II are covered by set III.
- March LR is similar to set III.
- March RAW covers other algorithms detecting dynamic faults.
- Ham_Walk is the only algorithm which contains a characteristic read-after-read operation and detects many unique faults.

Hence, a test set which is able to detect many different faults and is irredundant would consist of March LR, March RAW and Ham_Walk. This combination of algorithms would be expected to be most efficient.

6.4.4 Consistencies and Inconsistencies

Two independent ways have been used to classify the test algorithms. So, there may be differences in the results between the expected results based the analysis of march sequences and characteristic march elements, and the results of the empirical analysis of fault coverage and union and intersection. In this section, consistencies and inconsistencies between the two approaches are shown.

The structure of Algorithm B, March U, March LR and March SR are similar (see Table 6.3) and so are their test results (see Fig. 6.7). The expectation to have a similar performance and fault coverage due to the similar structure of those algorithms could successfully be shown by the experimental results. It is remarkable that Algorithm B shows a similar performance as recent algorithms although Algorithm B [18] has been developed long before functional fault models and fault primitives have systematically been developed which are base for the development of recent algorithms like March LR and March SR.

March AB and March AB1 have been designed to cover the same set of dynamic faults as March RAW and March RAW1 [8]. So, the performance and fault coverage of those algorithms has been compared and it could be shown that the performance of March RAW and March AB is very similar. The biggest part of faults is detected by both algorithms equally (see Fig. 6.10). However, the results of the comparison of March RAW1 and March AB1 are different. There is not a high similarity in their overall performance and fault coverage. So, according to [8], if the set of dynamic faults detected by both algorithms is the same, there are also many faults detected by March RAW1 or March AB1 individually. This means that March AB1 cannot replace March RAW1 or vice versa to detect more than only single-cell dynamic faults.

The most obvious inconsistency between theory and practice becomes visible at the comparison of SCAN and SCAN+. Due to the fact that the structure of SCAN is even a proper subset of that of SCAN+, it should be expected that also the set of faults detected by SCAN is a proper subset of set of faults detected by SCAN+. However, there are faults that are detected by SCAN but not by SCAN+ (see Fig. 6.8). This is highly remarkable but unfortunately cannot be explained with the methods used in this project. A specific analysis of single faulty devices that show the explained behavior would be necessary and may help to find the reasons for that faults that are detected only by SCAN.

6.5 Estimation of Fault Distribution

To estimate the distribution of different fault models within the test results, a classification of algorithms and corresponding faults models is necessary. Afterwards, the results can be analyzed and the distribution of faults can be estimated.

The sets of functional fault models from I to V differ in only one additional FFM from one set to the next. So, the fault estimation is done by evaluating the difference in the fault coverage from one set to the next. Beginning with the set of all faults, those faults are determined that are only detected by the algorithms of set I. Now the number of simple static single-cell faults is known. This set is now subtracted from the total set of faults, and the remaining set of faults is now without simple static single-cell faults. By repeated subtraction of one set after the other, the distribution of the four supersets of FFMs can be estimated within the test results [49].

Let Set 0 be the set of all algorithms (i.e. set I through V of Table 6.6) and let F_0 be the whole set of faults models, then F_0 contains 100% of faults.

Starting with set 0 and F_0, set I is subtracted from set 0, and at the same time the set of static simple single-cell faults (SSs) is subtracted from F_0. The remaining set of faults is then F_I, which contains any faults but no static simple single-cell faults.

$$0 - I \;\Rightarrow\; F_I = F_0 - \{SS\} \tag{30}$$

In general, the formula to derive the cardinality of each type of functional fault models is:

$$|\{FFM\}| = |F_n| - |F_{n+1}| \tag{31}$$

And so, the cardinality of SS is:

$$|\{SS\}| = |F_0| - |F_I| \tag{32}$$

Hence, the quantity of simple static single-cell faults could be determined. Based on the productive test results, the following values could be derived:

$$|F_0| = 2439 \quad \text{and} \quad |F_I| = 1610$$
$$|\{SS\}| = |F_0| - |F_I| = 829 \tag{33}$$

That means that 1610 of 2439 faults that have been detected, are simple static single cell faults, which are 34% of all faults. This process is repeated for all supersets of functional fault models until set V to estimate the whole fault distribution.

In the way described above, the cardinality of FFMs for each set has been determined for TN6531. After repeated subtraction, the following values have been derived: for each set: $|F_n|$ and $|\{FFM\}|$. The calculations are given in Appendix B and the results are summarized in Table 6.8.

Table 6.8. Determination of fault distribution

N	$\|F_n\|$		$\|\{FFM\}\|$		Percentage
0	2439				
I	1610	⇒	$\|\{SS\}\| = 829$	⇒	34%
II	1073	⇒	$\|\{\text{some CF}\}\| = 537$	⇒	22%
III	390	⇒	$\|\{\text{remaining CF}\}\| = 683$	⇒	28%
IV	171	⇒	$\|\{LF\}\| = 219$	⇒	9%
V	0	⇒	$\|\{DF\}\| = 171$	⇒	7%

According to (31), the cardinality and distribution of FFMs is as given in Table 6.9 and illustrated in Fig. 6.13.

Table 6.9. Fault distribution

FFMs	Percentage
simple static single cell faults (SS)	34%
some coupling faults (CF)	22%
remaining coupling faults (CF)	28%
linked faults (LF)	9%
dynamic faults (DF)	7%

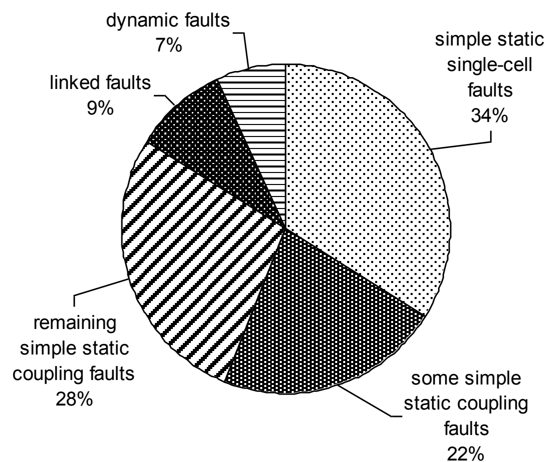

Figure 6.13. Fault distribution

The results are for TN6531, i.e. for 2439 faults in total and a test that took place after Burn-In. 34% of the faults are

assumed to be simple static single-cell faults and 50% (22% and 28%) of the faults are assumed to be simple static coupling faults. At least 9% are linked and 7% are dynamic faults.

This estimation of fault distribution is hardly comparable to any other results. On the one hand, no comparable results of a similar research are available, and on the other hand, the results are related to only one product and only one series of tests. Anyhow, the enormous number of simple faults is impressing due to the fact that the test was performed after wafer test. Obviously a lot of fault slipped through wafer test or occurred later due to Burn-In, although due to the previous wafer test it should be expected that most of the simple faults have already been detected, as during wafer testing, the RESET configuration of MBISTPLUS, which contains the algorithms SCAN and March C+, should have detected many simple static single-cell faults.

6.6 Summary and Conclusions

An effective combination of memory test algorithms is essential for productive memory test sets. Redundant testing should be avoided and a maximum of fault coverage should be achieved. So, the algorithms used in this study are analyzed on efficient combinations. To determine the efficiency of pairs of test algorithms, a factor Q_{Eff} is derived, which is a degree for efficiency of a combination of two test algorithms. A comparison of Q_{Eff} for each combination of test algorithms has been done and efficient and inefficient combinations could be determined.

The analysis of Q_{Eff}, as well as a deterministic analysis of the structure of march elements and test algorithms is used to classify similar test algorithms. Characteristic march sequences that are allocated to specific fault models have been identified and corresponding test algorithms are grouped into sets. Especially march sequences related to simple coupling

faults, linked faults and dynamic faults are of special interest. In addition to this deterministic method, the analysis of Q_{Eff}, and sub- and supersets of fault coverage is used to establish sets of similar algorithms. Algorithms with similar fault coverage are allocated to similar properties, and so to different functional fault models. Five sets of algorithms are defined which are allocated to specific sets of functional faults. Four algorithms seem to be of special interest for test sets: March RAW, March G, Ham5R and Ham_Walk. These algorithms are remarkable during the analysis of sub- and supersets. Obviously these algorithms are playing an important role for efficient test set generation. However, an optimization that leads to the minimal set of test algorithms is not possible with the previous method. The formal optimization and results of test set minimization will be presented in the following chapter.

Based on this classification, the selection of algorithms for test sets can be efficient, as the use of similar algorithms, and so, redundant testing, can be avoided.

As an application of classifying test algorithms, the distribution of faults within the test results of this project is estimated. Due to the repeated subtraction, the number of faults that are detected by a class of algorithms could be determined, and the functional fault modes could be allocated. It was determined that about one third of faults are simple static single cell faults, and half of the faults are simple static coupling faults. The remaining faults are either linked or dynamic faults.

Chapter 7
Test Set Optimization

Test set optimization means that the selection of tests for a productive memory test set should fulfill two requirements at the same time: maximum fault coverage and minimum test time. This means that from the whole set of test algorithms a minimal subset of algorithms is needed, which is able to detect all faults but the test effort should be as short as possible. That means that for production, the set of algorithms should be at a minimum. Therefore a formal optimization is needed, based on productive test results to determine the set of essential test algorithms.

In this chapter, the formal optimization and the set of essential algorithms as a result of the optimization are presented. Furthermore, based on the set of essential algorithms, a function of fault coverage over test length is derived that, depending on a desired yield, allows selecting test algorithms and estimating the minimal necessary test length.

7.1 Formal Optimization

The maximum fault coverage is surely achieved with the maximum set of algorithms. This means the whole study test set is applied to the memories. Then it can be assumed that the maximum of faults is detected. Now, optimization means to minimize the set of algorithms but keep the maximum fault coverage. The minimization is achieved with help of the well known ESPRESSO minimization algorithm [50] – [54].

ESPRESSO is a heuristic logic minimization algorithm developed to optimize digital logic gate circuits. However, the memory test results can be stated in such a way that ESPRESSO can be applied to optimize the memory test set. A brief description of ESPRESSO is given in Appendix C.

7.2 Test Data Preparation

The memory test results have to be prepared to be processed with ESPRESSO. Therefore a system is assumed representing the algorithms as inputs and the faults detected by a test as outputs (Fig. 7.1). The number of inputs to the system is m, the number of outputs is n.

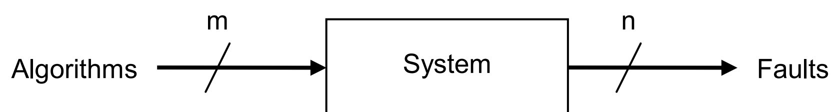

Figure 7.1. System representing algorithms and faults

The number of algorithms for each test is given by the study test set (Table 4.3). So, in this study, m is 30, and the algorithms (A_j) are indicated by their numbering given in

Table 4.3; e.g. SCAN is represented by A_1. The number of faults varies with the tests performed at different test numbers; e.g. n is 2439 for TN6531 (see Table 5.1).

The behavior of the system can be determined from the test results and is represented as a truth table. The format of this table is shown in Table 7.1.

Table 7.1. Algorithm-Fault truth table

Fail #	Fail	SCAN	SCAN+	MATS	MATS+	…	…	…	March G	Ham_Wk
i	F_i	A_1	A_2	A_3	A_4	…	A_j	…	A_{29}	A_{30}
1	1	-	-	1	-	…	…	…	-	-
2	1	1	1	1	1	…	…	…	1	1
3	1	-	-	1	1	…	…	…	1	1
4	1	1	1	1	1	…	…	…	1	1
5	1	-	-	-	1	…	…	…	1	1
…	…	…	…	…	…	…	…	…	…	…
n	…	…	…	…	…	…	…	…	…	…

Each fault F_i ($1 \le i \le n$) is represented by one row of the table. A fault F_i is deemed to be detected, if at least one algorithm A_j ($1 \le j \le 30$) detected it. So, for each fault F_i is: if one of the inputs A_j is true, F_i is true.

$$F_i = A_1 \vee A_2 \vee \cdots \vee A_j \vee \cdots \vee A_{29} \vee A_{30} \qquad (34)$$

Note that Table 7.1 is only an excerpt of the whole representation of test results and explains the data representation. An '1' denotes that an algorithm has detected a fault (i.e. input A_j is true), and '-' denotes that an algorithm A_j did not detect the fault; i.e. input A_j is 'Don't Care'. For all entries in the table, the output F_i is always true, as of course faults that are listed have to be detected first. As shown in (34), the test results for each fault F_i are represented as disjunction

of the inputs. Hence, the whole truth table representing the test results is given in conjunctive normal form (CNF). For the optimal test set, all faults F_1 … F_n have to be detected, that means the faults the fault information is conjunct.

$$F_1 \wedge F_2 \wedge \cdots \wedge F_i \wedge \cdots \wedge F_{n-1} \wedge F_n \qquad (35)$$

To be used with ESPRESSO software [52], the input format of the truth table has to be given in disjunctive normal form (DNF). The conversion from CNF to DNF is simply a conversion of all input and output values from ‘1’ to ‘0’, resp. ‘0’ to ‘1’; don’t care values (‘-‘) remain.

The input and output format for the software processing of the test results is explained in Appendix C.

7.3 Test Results

For each test number, the memory test data have been prepared for ESPRESSO and an input file was created. After minimization, the essential algorithms could be selected from the ESPRESSO output data.

Moreover, the sequence of the essential algorithms could be ordered in such a way, that a “function” could be derived that represents the fault coverage as a function of test length. This ordering is interesting if a desired yield should be reached at productive testing. Depending on the results of the “function” a specific subset of essential algorithms could be selected, which is sufficient to reach a desired yield.

7.3.1 Essential Algorithms

The set of essential algorithms is the minimum necessary set of algorithms to achieve full fault coverage. The essential

algorithms are the immediate result of the ESPRESSO minimization. The most representative test results are again the results of TN6531. Hence, these results are covered in this section. The results of all other test numbers are given in Appendix D.

The input table of the test results for TN6531 is a truth table containing 2439 rows - one for each fault. The output file for TN6531 is short enough and is given in Fig. 7.2.

```
.i 30
.o 1
.p 4
-------1-----1--1111-1---11-11 1
-------1-----11-111--1---11-11 1
-------1-----1--1111-1-1--1-11 1
-------1-----11-111--1-1--1-11 1
.e
```

Figure 7.2. ESPRESSO output file of TN6531

The output table of ESPRESSO consists of four rows. So, there are four equivalent sets of essential algorithms. Exemplarily the fourth results (marked in Fig. 7.2) is evaluated. The choice of a set of essential algorithm depends on criteria like the number or complexity of test algorithms. The more different algorithms are used, the more additional time is needed for configuration. Due to a minimal number of algorithms, this configuration time can be minimized. Also, if March G can be avoided in a test set, the test time can be kept very low as no additional delay time between march elements is necessary. Findings for effectiveness of pairs of algorithms are playing also a role in the choice of a set of essential algorithms (see Chapter 6). All these factors have been taken into account for the selection of sets of essential algorithms for all test numbers. The results for each test at different test numbers that are given in Appendix D follow these criteria

and only one possible set of essential algorithms is evaluated per test number.

Each '1' in the output table shown in Fig. 7.2 denotes an essential algorithm. According to position j, algorithm A_j is essential. Eleven algorithms are forming the set of essential algorithms, which is highlighted in Table 7.2.

Table 7.2. Essential algorithms for TN6531

A_j	Algorithm	Test Length
A_8	March B	17n
A_{14}	March U	14n
A_{15}	March X	6n
A_{17}	March LR	14n
A_{18}	March LA	22n
A_{19}	March RAW	26n
A_{22}	March AB1	11n
A_{24}	March SR	14n
A_{27}	Ham5R	25n
A_{29}	March G	24n + 2*D*
A_{30}	Ham_Walk	15n

March LR, March RAW and Ham_Walk have already been determined as efficient test algorithms. The results of classifying algorithms presented in Chapter 6 show that due to statistical analysis, these algorithms are of high interest for efficient test sets. This finding is now confirmed by determining these algorithms as essential.

In the same manner as presented for TN6531, the test results of each test number have been optimized, and for each test number not only one set of essential algorithms was derived, but several sets of essential algorithms are possible.

The length of the set derived for TN6531 is 187n + 2*D*. This is the minimum test length to achieve full fault coverage with a set of eleven test algorithms. So, in this example optimal test set means the use of eleven algorithms with a minimal

necessary test length of 187n + 2*D*. The test time for this set of essential algorithms is then (see formula 5):

$$TT_{6531} = \frac{187 \cdot 261.56\text{kB} \cdot \frac{1024 \cdot 8}{32\text{kB}}}{180\text{MHz}} + 2 \cdot 100\text{ms} = 269.6\text{ms} \quad (36)$$

7.3.2 Fault Coverage related to Test Length

Not only 100% fault coverage at minimal test length is necessary in productive memory testing, also less fault coverage may be acceptable if products are not highly safety critical or error correction is used. An important question was: "Can fault coverage be derived from test length and vice versa?"

To answer this question, the essential algorithms can be taken and ordered in such way that the fault coverage (FC) is interpreted as function of the test length (TL).

$$\text{FC} = f(\text{TL}) \quad (37)$$

Of course this is not a mathematical function, as there is no rule that enables to derive the fault coverage from the test length. However, the results of the memory tests can be taken to illustrate the fault coverage over test length. For the allocation, the set of essential algorithms is taken, as this set is the minimum to achieve 100% fault coverage. The ordering of essential algorithms follows the weighting:

1. maximum increase of fault coverage (maximum ΔFC)
2. minimum increase of test length (minimum ΔTL)

Hence, the first algorithm to be taken is the one with highest fault coverage, second, the one which increases the total fault coverage of the set most. If two or more algorithms are equivalent concerning ΔFC, the shortest one is taken. If

two or more algorithms are of same length, the selection is arbitrary. According to the set of essential algorithms at TN6531 and the productive test results, the following data could be derived and listed in Table 7.3. Note that in this case, for ΔFC, the absolute number of difference of faults is given instead of a percentage, as the comparison of these values is easier than that of percentage.

Table 7.3. Fault coverage over test length

#	Algorithm	ΔFC	ΔTL	FC	TL
1	Ham_Walk	2063	15n	84.58%	15n
2	March RAW	284	26n	96.23%	41n
3	Ham5R	36	25n	97.70%	66n
4	March G	30	23n+2*D*	98.93%	89n+2*D*
5	March AB1	13	11n	99.47%	100n+2*D*
6	March LR	5	14n	99.67%	114n+2*D*
7	March X	3	6n	99.79%	120n+2*D*
8	March LA	2	22n	99.88%	142n+2*D*
9	March U	1	14n	99.92%	156n+2*D*
10	March B	1	14n	99.96%	170n+2*D*
11	March SR	1	17n	100%	187n+2*D*

The results of Table 7.3 are illustrated in Fig 7.3. The fault coverage is plotted against test length.

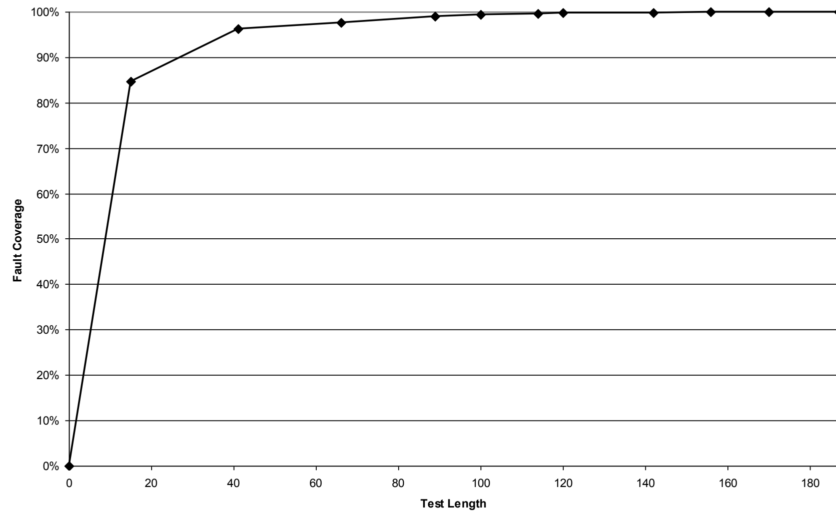

Figure 7.3. Fault coverage over test length

Note that the graphical illustration in Fig. 7.3 depends on discrete values at discrete test length. The single points are only connected to a continuous graph to point out the trend of fault coverage over test length.

The weighted sequence of essential algorithms allows a selection of a subset to achieve a desired yield. One can see that the fault coverage increases very fast already at low test length. After applying three algorithms (Ham_Walk, March RAW and Ham5R) with length of 66n, a fault coverage of already 97.7% is reached.

The test time for this reduced set of essential algorithms is:

$$TT_{6531_reduced} = \frac{66 \cdot 261.56\text{kB} \cdot \frac{1024 \cdot 8}{32\text{kB}}}{180\text{MHz}} = 24.6\text{ms} \qquad (38)$$

In contrast, the effort for 100% fault coverage is relatively high (11 algorithms and 269.6ms test time). With increasing test length the increase of additionally covered faults (ΔFC) decreases dramatically. Each of the last three algorithms in this ordering detects only one additional fault. These seem to

be the most interesting faults as they are uniquely detected and are of special interest for a specific fault analysis. Depending on the requirements on safety of products and hence on the specification of the productive test flow a 100% fault coverage may not be necessary. Remaining faults could be detected and corrected during operation by ECC methods.

It turns out that very high fault coverage is already achieved with relatively low effort. Although the essential set consists of eleven test algorithms, a few of them are sufficient to reach a rather high yield.

For all other test number, the same analysis is done and can be found in Appendix D. Especially algorithms March RAW, Ham_Walk and Ham5R are most important for all tests. At least one of these algorithms appears for each test number, and furthermore these algorithms are on top of the list of essential algorithms, and hence, are most important for high fault coverage. This is a highly remarkable fact, as these algorithms already have been identified as outstanding during analysis of sub- and supersets in Chapter 6. The assumption that these algorithms are important in test set generation is confirmed by the results in this chapter.

So, independent of environmental test conditions, three algorithms could be identified that appear in each set of essential algorithms and that are most important for a high fault coverage of the optimal test set.

This analysis and especially the ordering of test algorithms are interesting for tests with highly restricted test time. Depending on the test length, the expected yield can be estimated, and vice versa. A selection of test algorithms can now be done with respect to both, test length and desired yield. Depending on the properties and objectives of a specific test, an optimal proportion of test length and fault coverage can be selected.

7.4 Summary and Conclusions

From the whole set of test and fail data, the essential test algorithms are determined that represent a set of test algorithms to achieve full fault coverage at a minimum test effort at the same time. By treating the test results as the representation of a logic system that uses the test algorithms as inputs and the fail information as output, the heuristic logic minimization algorithm ESPRESSO could be used to optimize the test set and to derive the essential algorithms. For all test conditions, more than one possible set of essential test algorithms has been derived. To choose which solution to use, criteria like test length, no delay time or effective pairs of algorithms within the set are relevant. For all test numbers, the most important algorithms are March RAW, Ham5W and Ham_Walk.

The essential algorithms of a test could also be ordered in such a way that the fault coverage could be represented as a "function" of test length. This is a necessary requirement to select algorithms for a productive memory test set. If a specific yield is desired, the necessary number of test algorithms can be kept to a minimum and so the test time can be shortened. Exactly those three algorithms that are most essential (March RAW, Ham5R and Ham_Walk) are sufficient to achieve about 98% fault coverage for TN6531, and for all test numbers, at least one of the algorithms March RAW, Ham5R or Ham_Walk is included in the set of essential algorithms, however, mainly all of them. Furthermore, these algorithms are most important for the optimal test sets, as, if the algorithms are ordered, these algorithms are on top of the list detecting the major part of faults and hence improve the total fault coverage best. These algorithms are exactly those that have also been identified as important and outstanding during the analysis of sub- and supersets in the previous chapter.

Depending on the position of a test stage within the whole productive test flow, this might me sufficient as the remaining faults are covered by a later test in the flow or are treated by ECC methods. By ordering the algorithms the function relates test length and fault coverage, and depending on a desired yield, the necessary set of algorithms can be picked.

Chapter 8
Variation of Fault Manifestation

Memory faults vary during life time. Previous investigations on embedded memories of single devices at Infineon have shown that the behavior of some faults is strange. Some faults only appear at certain circumstances, e.g. only during functional use of the device. However, these faults cannot be reproduced by any test during comprehensive investigations.

This is a reason to carry out an analysis on the behavior and variation of faults [55, 56]. The test setup, which is used in this project, enables the analysis of fault variation during Burn-In and to analyze the behavior of faults before and after stresses. In this chapter the variation of fault manifestation during Burn-In is analyzed, and productive test results are presented that show this fault variation.

8.1 Setup and Environment

To obtain comparable test results and to see the effects of Burn-In, the environmental test conditions before and after Burn-In stress should be the same. The environmental test conditions are defined by the productive test flow, and there are two tests before and after Burn-In with similar environmental test conditions: TN1522, TN1622 and TN6531 and TN6631. From the total test flow (Fig. 4.6), the relevant part is shown in Fig 8.1.

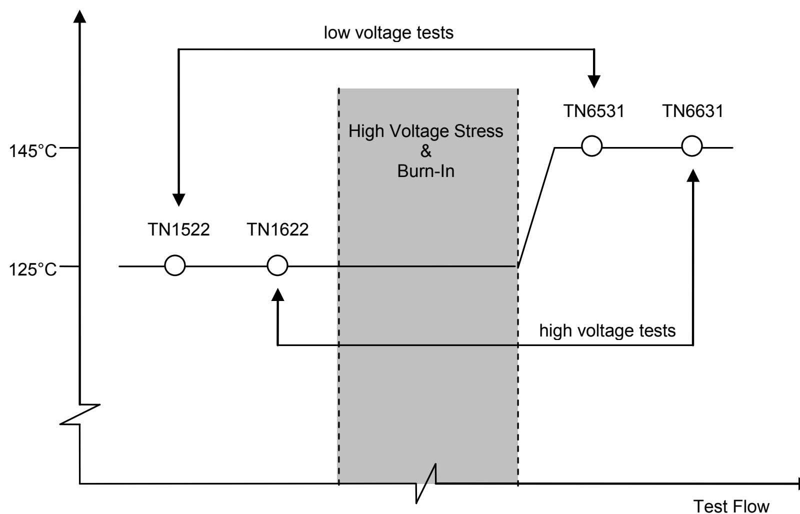

Figure 8.1. Test flow surrounding Burn-In

Two pairs of test are compared: one with low and one with high supply voltage. The results of TN1522 are compared to those of TN6531, and the results of TN1622 are compared to TN6631. The difference of temperature before and after Burn-In could not be avoided due to the predefined test flow. Hence, an influence of temperature variation can not entirely be excluded. However, these are the only tests for a comparison, and the difference in temperature is acceptable.

The analysis is done for low voltage and high voltage test results separately in order to recognize if the supply voltage has additional effects during Burn-In. The high voltage stress and Burn-In is performed at a temperature of 125°C, and the duration of Burn-In is 12 hours in this study.

Two kinds of test results are analyzed [55, 56]:

- Increase of fault coverage due to Burn-In.
- Variation of faulty behavior due to Burn-In.

Both data are examined for low and high voltage tests separately.

8.2 Increase of Fault Coverage

The test results for low voltage (TN1522 and TN6531) and high voltage (TN1622 and TN6631) are summarized in Table 8.1.

Table 8.1. Fault coverage during Burn-In

	Test Number	# of Faults
low voltage	1522 (before BI)	617
	6531 (after BI)	2439
	1522 ∩ 6531	585
	1522 ∪ 6531	2471
high voltage	1622 (before BI)	56
	6631 (after BI)	175
	1622 ∩ 6631	43
	1622 ∪ 6631	188

The results show the number of faults detected before and after Burn-In, as well as intersection and union of these sets. The distribution of faults before and after Burn-In is illustrated in Fig 8.2 for low voltage (TN1522 and TN6531), and in Fig 8.3. for high voltage (TN1622 and TN6631).

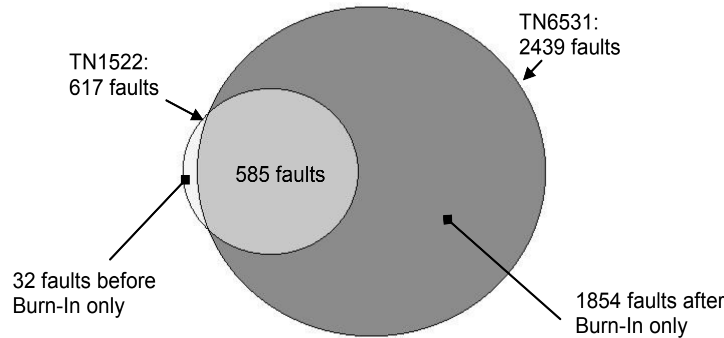

Figure 8.2. Fault distribution at low voltage

The comparison of fault coverage before and after Burn-In in Fig 8.2. shows impressively a four-times-increase from 617 before to 2439 faults after Burn-In. 1854 faults have not been detected before Burn-In, and 585 faults are detected before and after Burn-In equally. A very small part of 32 faults only occurs before Burn-In. These faults are healed due to stress and are no longer detectable afterwards. This effect could not be analyzed in more detail, as the test data in this project are not detailed enough for a specific faults analysis.

The high increase of faults could be expected. The characteristic of Burn-In is artificial aging, which is designed to activate latent faults that would appear during the course of life time of a product (see Bathtub-curve in Fig. 4.3) [41, 57]. After all, about 75% (1854 of 2471) of the faults that are treated in this analysis are only detected due to Burn-In and high voltage stress. With regard to these results, the effect of Burn-In is enormous and cannot be ignored for productive memory testing – especially for highly safety critical products. Likewise, the results for high voltage are shown in Fig 8.3.

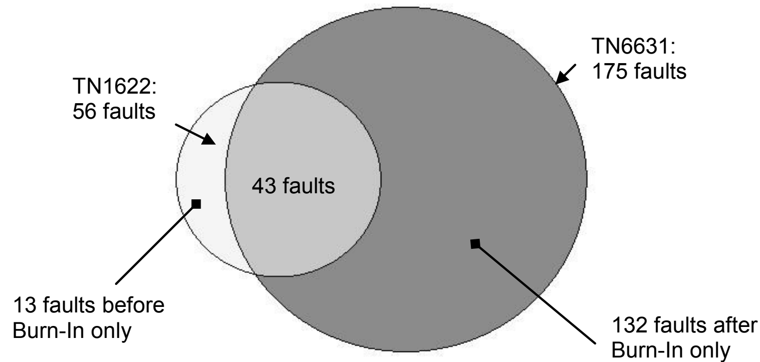

Figure 8.3. Fault distribution at high voltage

The same effects as with low supply voltage could be recognized at high voltage tests, even though the total number of faults is much lower. Hence, the results are less representative, but here also, the number of faults increases from 56 faults before to 175 faults after Burn-In. This is a more than three-times-increase. However, on base of 188 faults no meaningful and representative statistical analysis would be reasonable.

8.3 Variation of Fault Behavior

The results for low and high voltage tests show that the major part of Pre-Burn-In faults remains during Burn-In. These faults are of special interest concerning the variation of faulty behavior. In the following sections, it will be shown that Burn-In influences the manifestation of these faults, i.e. the fault model representing the fault changes due to stress, and the same fault shows up with a different behavior.

8.3.1 Data Evaluation Technique

From the pure test results, the faulty behavior, i.e. the fault model, cannot be identified. But with help of the results for classification of algorithms and faults, presented in Chapter 6, the appearance of specific fault models can be estimated based on the fault coverage of specific test algorithms. This classification is used to distinguish between three kinds of faults:

- static faults
- coupling faults
- dynamic faults

To achieve the differentiation, three test algorithms are chosen that mainly detect the three types of faults. The algorithms are March C-, March LR and March RAW. The allocation of these algorithms and corresponding faults models are given in Table 8.2.

Table 8.2. Algorithm-fault-allocation

Algorithm	predominantly corresponding fault models
March C-	static faults
March LR	static faults coupling faults
March RAW	static faults coupling faults dynamic faults

The basic fault model is determined by analyzing the coverage of a fault. Dynamic faults are only covered by March RAW, coupling faults are additionally covered by March LR, and static faults are covered by any of the three test algorithms. Hence, if a fault is only detected by March RAW, it can be assumed to be dynamic. If a fault is detected by March LR and March RAW, but not by March C- it is assumed

to be coupling. And if a fault is detected by all three test algorithms, it is assumed to be static. Likewise to the classification of faults and algorithms, the distribution of faults is determined by repeated subtraction of sets of faults.

This analysis is done only for those faults that are remaining after Burn-In, i.e. that have been present before and still appear after Burn-In. These faults are represented by the intersection in Figs. 8.2 and 8.3. For the low voltage test series, these are 585 faults, for high voltage tests there are 43 faults.

8.3.2 Test Results

For low voltage tests, the following distribution of fault models could be determined before and after Burn-In.

Table 8.3. Fault model distribution at low voltage

Fault Model	before Burn-In		after Burn-In	
	|F|	percentage	|F|	percentage
static faults	126	52,0%	233	31,8%
coupling faults	155	26,5%	166	28,4%
dynamic faults	304	21,5%	186	39,8%
Total	585	100%	585	100%

The results for the low voltage test, i.e. at TN1522 before and at TN6531 after Burn-In, show that the distribution of static, coupling and linked faults varies before and after Burn-In. From totally 585 faults, 126 faults are static before Burn-In and 233 faults are static after Burn-In. To the same extent the number of dynamic faults decreases from 304 before to 186 faults afterwards. The number of coupling faults remains roughly at 155 resp. 166 faults. Due to the detailed information gathered during the tests, the faults could explicitly be identified, and the results before and after Burn-In refer exactly to the same set of faults.

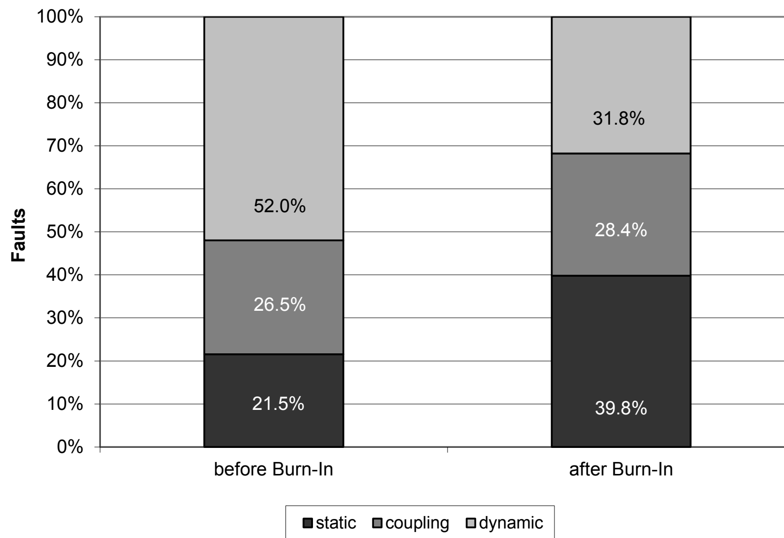

Figure 8.4. Fault model distribution a low voltage

The high voltage stress and Burn-In between the two tests influences some faults to change their appearance from dynamic to static. Dynamic means, the faults are latent before, and static means they are stable after Burn-In. The stress due to Burn-In causes these latent faults to manifest. This variation in faulty behavior is observable for about 20% of faults at low voltage test.

The distribution of faults at high voltage test, i.e. at TN1622 before and TN6631 after Burn-In is similar to those of low voltage. Table 8.4 summarizes the results for high voltage.

Table 8.4. Fault model distribution at high voltage

Fault Model	before Burn-In		after Burn-In	
	|F|	percentage	|F|	percentage
static faults	18	44,1%	24	32,6%
coupling faults	6	14,0%	5	11,6%
dynamic faults	19	41,9%	14	55,8%
Total	43	100%		100%

The variation of dynamic and static faults at high supply voltage is similar to that at low voltage. However the total number of faults in the analysis for high voltage is much lower. So, the results are less representative, but a variation is nevertheless observable. The results of Table 8.4 are illustrated in Fig 8.5.

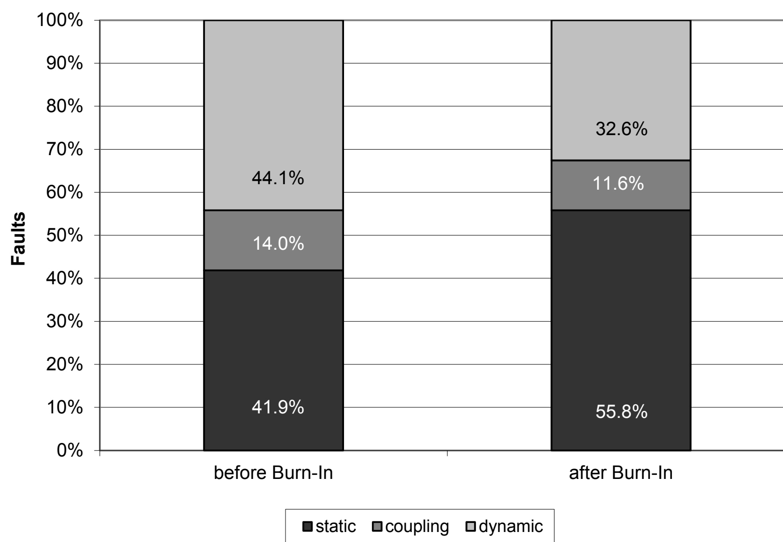

Figure 8.5. Fault model distribution at high voltage

This finding shows that not only new faults appear after Burn-In, but also existing faults vary their behavior. This is important if different test sets should be used before (e.g. at wafer test) and after Burn-In.

Also the quality of dynamic faults can be rated. The faults that have changed their behavior from dynamic into static are less stable than others. So the physical defects that cause these faults are possibly different from that of stable dynamic faults. A closer analysis of the physical defects could answer the question on reasons for the fault variation.

8.4 Summary and Conclusions

Burn-In is a process of quality assurance during memory testing. Artificial aging due to high temperature and high voltage stress makes latent faults to become visible and influences their faulty behavior. Full test sets haven been performed at comparable environmental conditions before and after Burn-In stress, and the test results have been compared. Both, for lowered and raised supply voltage, the number of detectable faults increased during Burn-In. The number of faults increases by a factor of about four at low voltage, and about three at high voltage. Without Burn-In these additional faults would have remained undetected. Especially for highly safety critical products – as used in this project – Burn-In is indispensable to ensure memory quality.

Besides the increase of detectable faults, also the manifestation of some faults have been present before Burn-In and are still observable after Burn-In could be observed. Based on classification of faults and memory test algorithms, those faults that have been present before and after Burn-In have been analyzed on their behavior. Static, coupling and dynamic faults have been distinguished and the analysis has shown that faults that have been dynamic before Burn-In became static afterwards. About 20% faults are affected at lowered supply voltage and about 10% at increased supply voltage. This shows that faults are variable. The behavior of one fault is not fixed but it may change under environmental conditions and so, also the detectability varies. As the effort to detect dynamic faults is normally higher as they are latent, static faults are much easier to detect. This finding may influence the selection of test sets for different environmental conditions and for different stages in the test flow (before and after Burn-In), as different types of faults appear.

Chapter 9
Perspectives

After all, basics on memory test, algorithms and faults, test setup and data acquisition, the analysis on effectiveness of test algorithms, test set optimization and fault analysis during Burn-In, this chapter is dedicated to future work. Not all questions could be answered in this work and many more questions arose during the project.

One intention of the project was to run the memory tests on products of different technology, 130nm and 90nm Unfortunately, this could not be realized in the timeframe of this project, as the data acquisition using MBISTPLUS V4.2 is much more complex than originally supposed, and the existing test programs could not handle the enormous number of test data. Moreover the throughput of 90nm products was not yet high enough to obtain enough test results for a statistical analysis. So, a future task should be to repeat the statistical analysis with an extended test set on 90nm products using MBISTPLUS V4.2., and the following analyses should be done:

- Comparing the test results of different technologies (130nm and 90nm), and analyze the impact of technological differences on fault coverage and failure classes [58]; especially the development of dynamic faults as a results of shrinking technology.

- Analyze the test results of additional test algorithms performed with MBISTPLUS V4.2.; especially the fault coverage of complex test algorithms GAL5 and GAL9.
- Analyze the impact of additional addressing modes.
- Analyze the impact of self timing parameters read timing, write timing and weak write driver, and determine the optimal setting of these parameters for optimal yield.

All these analyses need the same background of a high amount of test results that guarantee a meaningful statistical analysis. With respect to dynamic faults, the influence of different delay time used with March G could also be analyzed. March G could be used with different settings of delay time and the variation of fault coverage could be observed. If there is a variation, it clearly depends on the variation of delay time. The faults that are only related to delay time are data retention faults and for the detection of these faults, the optimal setting of delay time could be determined.

The results of this project are based on a large number of test data. However, the density of details in these data suffers from this fact. For a more detailed analysis, the productive test flow is inappropriate. Instead, single devices could be picked from the flow and analyzed manually. In this case, the classification of fault models and test algorithms could be more accurate, as single faulty read operations could be identified, and so the corresponding fault primitives.

This project provides a large number of productive test results, which allow determining if a fault is detected by a certain test algorithm or not. This information could also be used for automatic test generation. In [8, 27, 28], the test algorithms are generated based on simulation results. The use of real test results may improve the automated test generation and so would increase the effectiveness of test algorithms. A

future project could bring these two things together and would allow generating memory test algorithms based on real, productive test data.

Picking defect devices from the productive test flow means very high effort and is nearly impossible as the fail analysis could only take place after a certain number of devices has been tested to achieve a statistical meaningful number. So, effects that occur only with a few devices do not become visible immediately. Such faults could be single unique faults that occur only with some algorithms or faults that can only be detected by special and specific algorithms. However, such devices should be picked and analyzed in more detail for a more precise fault analysis. The results of this work help to identify such typical faults and the circumstances that make these faults detectable. If a comprehensive test set is used during ramp-up to optimize the selection of test algorithms and conditions, also an automated fault analysis could be added. Devices that contain unique faults or faults that occur only with specific test conditions and algorithms could be marked separately and picked. So the findings of a previous statistical analysis could be used to identify noticeable faults and to take them for a detailed analysis. If this process is done repeatedly, a database would grow that allows on-line automated fault analysis, also during productive testing, if single devices cannot be picked from the test flow.

References

1 van de Goor, A.J., 1998. *Testing Semiconductor Memories: theory and practice.* ComTex Publishing, Gouda, The Netherlands

2 Hamdioui, S., 2004. *Testing static random access memories: Defects, fault models and test patterns.* Kluwer Academic Publishers, Dordrecht, The Netherlands

3 Hamdioui, S., Al-Ars, Z., van de Goor, A.J., Rodgers, M., 2004. Linked Faults in Random Access Memories: Concept, Fault Models, Test Algorithms and Industrial Results. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.* 23(5), pp. 737-757

4 van de Goor, A.J., Gaydadjiev, G.N., 1997. March U: a test for unlinked memory faults. *In Proc.: Circuits, Devices and Systems.* pp. 155-160

5 Hamdioui, S., van de Goor, A.J., Reyes, J.D., Rodgers, M., 2006. Memory Test Experiment: Industrial Results and Data. *Computers and Digital Techniques.* 153(1), pp. 1-8

6 Al-Ars, Z., Hamdioui, S., Gaydadjiev, G.N., Vissiliadis. S., 2008. Test Set Development for Cache Memories in Modern Microprocessors. *IEEE Transactions on VLSI Systems,* 16(6). pp. 725-732

7 Hamdioui, S., Al-Ars, Z., van de Goor, A.J., 2002. Testing Static and Dynamic Faults in Random Access Memories. *In Proc.: 20th IEEE VLSI Test Symposium.* pp. 395-400

8 Benso, A., Bosio, A., Di Carlo, S., Di Natale, G., Prinetto, P., 2005. March AB, March AB1: New March Tests for Unlinked Dynamic Memory Faults. *In Proc.: IEEE International Test Conference.* pp. 834-841

9 Hamdioui, S., van de Goor, A.J., Rodgers, M., 2002. March SS: A Test for All Static Simple RAM Faults. *In Proc.: Int. Memory Technology, Design and Testing.* pp. 95-100

10 Adams, R.D., 2003. *High Performance Memory Testing: Design Principles, Fault Modeling and Self-Test.* Kluwer Academic Publishers, Dordrecht, The Netherlands

11 van de Goor, A.J., Al-Ars Z., 2000. Functional Memory Faults: A Formal Notation and a Taxonomy. *In Proc.: 18th IEEE VLSI Test Symposium.* pp. 281-289

12 Al-Ars, Z., Hamdioui, S., van de Goor, A.J., 2003. A fault primitive based analysis of linked faults in RAMs. *Records of IEEE Int. Workshop on Memory Technology, Design and Testing.* pp. 33-39

13 van de Goor, A.J., Hamdioui, S., Wadsworth, R., 2004. Detecting Faults in the Peripheral Circuits and an Evaluation of SRAM Tests. *In Proc.: IEEE International Test Conference.* pp. 114-123

14 Offerman, A., van de Goor, A.J., 1997. An open notation for Memory Tests. *In Proc.: Int. Workshop on Memory Technology, Design and Testing.* pp. 71-78

15 Knaizuk, J., Hartmann, C.R.P., 1977. An Optimal Algorithm for Testing Stuck-At Faults in Random Access Memories. *IEEE Transactions on Computers,* C-26(11). pp. 1141-1144

16 Nair, R., 1979. Comments on 'An Optimal Algorithm for Testing Stuck-At Faults in Random Access Memories'. *IEEE Transactions on Computers*, C-28(3). pp. 258-261

17 Suk, D.S., Reddy, S.M., 1981. A March Test for Functional Fault in Semiconductor Random Access Memories. *IEEE Transactions on Computers,* C-30(12). pp. 982-985

18 Marinescu, M., 1982. Simple and Efficient Algorithms for Functional RAM Testing. *IEEE International Test Conference.* pp. 236-239

19 van de Goor, A.J., Gaydadjiev, G.N., Yarmolik, V.N., Mikitjuk, V.G., 1996. March LR: A Test for Realistic Linked Faults. *In Proc.: 14th VLSI Test Symposium.* pp. 272-280

20 Hamdioui, S., Al-Ars, Z., van de Goor, A.J., Rodgers, M., 2003. Dynamic Faults in Random-Access Memories: Concept, Fault Models and Tests. *Journal of Electronic Testing: Theory and Applications.* vol. 19(2), pp. 195-205

21 van de Goor, A.J., 2008. *Testing Memories: Fault Models, Algorithms, Tests and Industrial Results.* Unpublished

22 van de Goor, A.J., 2008. *Testing Memories: New Fault Models, Tests, DFT, BIST, BISR, Industrial Results; Flash Memories and Soft Errors.* Unpublished

23 de Jonge, J.H., Smeulders, A.J., 1976. Moving inversions test pattern in thorough, yet speedy. *Computer design.* pp. 169-173

24 Schanstra, I., van de Goor, A.J., 1999. Industrial Evaluation of Stress Combinations for March Tests Applied to SRAMs. *In Proc.: IEEE International Test Conference.* pp. 983-992

25 Breuer, M.A., Friedman, A.D., 1976. *Diagnosis and reliable design of digital systems.* Computer Sciences Press Inc., CA, USA

26 van de Goor, A.J., Gaydadjiev, G.N., Yarmolik, V.N., Mikitjuk, V.G., 1997. March LA: A Test for Linked Memory Faults. *In Proc.: IEEE European Design and Test Conference.* p. 627

27 Bosio, A., Di Carlo, S., Di Natale, G., Prinetto, P., 2007. March AB, a state-of-the-art march test for realistic static linked faults and dynamic faults in SRAMs. *IEEE Computer & Digital Techniques.* 1(3), pp. 237-245

28 Bosio, A., Di Natale, G., 2008. March Test BDN: A new March Test for Dynamic Faults. *IEEE Int. Conference on Automation, Quality and Testing, Robotics.* vol. 1, pp. 85-89

29 Hamdioui, S., van de Goor, A.J., 2000. An Experimental Analysis of Spot Defects in SRAMs: Realistic Fault Models and Tests. *In Proc.: 9th Asian Test Symposium.* pp. 131-138

30 Hamdioui, S., Al-Ars, Z., van de Goor, A.J., Rodgers, M., 2003. March SL: A Test for All Static Linked Memory Faults. *In Proc.: 12th Asian Test Symposium.* pp. 372-377

31 van de Goor, A.J., 1993. Using March Tests to Test SRAMs. *IEEE Design & Test of Computers.* 10(1), pp. 8-14

32 MBIST+ Generic Module – Internal Target Specification V3.0, *Infineon Technologies AG – internal document*

33 Hamdioui, S., Al-Ars, Z., van de Goor, A.J., Wadsworth, R., 2005. Impact of Stresses on the Fault Coverage of Memroy Tests. *In Proc.: IEEE Int. Workshop on Memory Technology, Design and Testing.* pp. 103-108

34 Hamdioui, S., Al-Ars, Z., van de Goor, A.J., 2006. Opens and Delay Faults in CMOS RAM Address Decoders. *IEEE Transactions on Computers.* 55(12), pp. 1630-1639

35 van de Goor, A.J., Hamdioui, S., Al-Ars, Z., 2004. Tests for Address Decoder Faults in RAMs due to Inter-gate Opens. *In Proc.: 9th IEEE European Test Symposium.* pp. 49-54

36 van de Goor, A.J., Schanstra, I., 2002. Address and Data Scrambling: Causes and Impact on Memroy Tests. *In Proc.:1st Int. Workshop on Electronic Design, Test and Applications.* pp. 128-136

37 Majhi, A., Azimane, M., Gronthoud, G., Lousberg, M., Eichenberger, S., Bowen, F., 2005. Memory Testing Improvements through Different Stress Conditions. *In Proc.: 31st European Solid-State Circuits Conference.* pp. 299-302

38 MBISTPLUS Generic Module – Internal Target Specification V4.2, *Infineon Technologies AG – internal document*

39 Vollertsen, R.P., 1999. Burn-In. *In Proc.: IEEE Int. Integrated Reliability Workshop.* pp. 167-173

40 Cockburn, B.F., 1994. Tutorial on Semiconductor Memory Testing. *Journal of Electronic Testing: Theory and Applications.* 5(4), pp. 321-336

41 Lorenz, D., Barke, M., Schlichmann, U., 2010. Aging analysis at gate and macro cell level. *In Proc.: IEEE Int. Conference on Computer Aided Design.* pp. 77-84

42 Infineon Technologies AG, 2009. TC1797 32-Bit Single-Chip Micorcontroller Data Sheet, V1.2 2009-09

43 Design for Analysis Specification AudoMax, *Infineon Technologies AG – internal document*

44 Kuhn, H., 2010. "Innovative Testlösungen für Automotive Halbleiter". *presented at 4. GMM/Gi/ITG-Fachtagung ‚Zuverlässigkeit und Entwurf', 13.9.-15.9.2010, Wildbad Kreuth, Germany*

45 M1747 IBIS Flow for V1010u2 Productive Testplan, *Infineon Technologies AG – internal document*

46 Dasdan, A., Hom, I., 2006. Handling Inverted Temperature Dependence in Static Timing Analysis. *ACM Transactions on Design Automation of Electronic Systems.* 11(2), pp. 306-324

47 Sharifkhani, M., Jahinuzzaman, S.M., Sachdev, M., 2006. Dynamic Data Stability in SRAM Cells and its Implications on Data Stability Tests. *In Proc.: IEEE Int. Workshop on Memory Technology, Design and Testing.* pp. 68-74

48 Hamdioui, S., Wadsworth, R., Reyes, J.D., van de Goor, A.J., 2003. Importance of Dynamic Faults for New SRAM Technologies. *In Proc.: 8th IEEE European Test Workshop.* pp. 29-34

49 Linder, M., Eder, A., Oberländer, K., Huch, M., 2011. Effectiveness of Memory Test Algorithms and Fault Distribution in SRAMs. *paper presented at ETS'11.* 23.5.-27.5.2011 , Trondheim, Norway

50 Brayton, R.K., Hachtel, G.D., McMullen, C.T., Sangiovanni-Vincentelli. A.L., 1984. *Logic Minimization Algorithms for VLSI Synthesis.* Kluwer Academic Publshers, Boston, USA

51 De Micheli, G., 1994. *Synthesis and optimization of digital circuits.* McGraw-Hill, New York, USA

52 *Logic Friday, Free Software for boolean logic optimization, analysis, and synthesis.* accessed 02/09/2010, http://www.sontrak.com/download/espresso.zip

53 Valparaiso Univerity. *ESPRESSO: Logic Minimization Software.* accessed: 02/09/2010, http://diamond.gem.valpo.edu/~dhart/ece110/espresso/tutorial.html

54 Chiusano, S., A short introduction to Espresso. accessed: 03/09/2010, http://www.uic.edu/classes/ece/ece465/06/tools/A%20short%20introduction%20to%20Espresso.pdf

55 Linder, M., Eder, A., Oberländer, K., Huch, M., 2011. Variations of Fault Manifestation during Burn-In: A case Study on Industrial SRAM Test Results. *In Proc.: 17th IEEE International On-Line Testing Symposium.* 13. - 15.7.2011, Athens, Greece. pp. 246-249

56 Linder, M., Eder, A., Oberländer K., Huch, M., 2011. Memory testing during Burn-In: Test Strategy and Experimental Results. *23. GI/GMM/ITG-Workshop 'Test methoden und Zuverlässigkeit von Schaltungen und Systemen'*. 27.2. - 1.3.2011, Passau, Germany. pp. 13-17

57 Wang, L., Ye, Q., Wong, R., Liehr, M., 2007. Product Burn-in Stress Impacts on SRAM Array Performance. *In Proc.: 45th IEEE Int. Reliability Physics Symposium.* pp. 666-667

58 Chen, Q., Mahmoodi, H., Bhunia, S., Roy, K., 2005. Modeling and Testing of SRAM for New Failure Mechanisms due to Process Variations in Nanoscale CMOS. *In Proc.: 23rd IEEE VLSI Test Symposium.* pp. 292-297

Additional Literature

59 Abadir, S.M., Reghbati, H.K., 1983. Functional Testing of Semiconductor Random Access Memories. *ACM Computing Surveys.* 15(3), pp. 175-198

60 Al-Ars, Z., Herzog, M., Schanstra, I., van de Goor, A.J., 2004. Influence of Bit Line Twisting on the Faulty Behavior of DRAMs. *Records of IEEE Int. Workshop on Memory Technology, Design and Testing.* pp. 32-37

61 Al-Ars, Z., van de Goor, A.J., 2004. Soft Faults and the Importance of Stresses in Memory Testing. *In Proc.: Design, Automation and Test in Europe.* vol. 2, pp. 1084-1089

62 Amerasekera, E.A., Najm, F.N., 1997. *Failure Mechanisms in Semiconductor Devices.* John Wiley & Sons, Chichester, England

63 Benso, A., Bosio, A., Di Carlo, S., Di Natale, G., Prinetto, P., 2006. A 22n March Test for Realistic Static Linked Faults in SRAMs. *In Proc.: 11th IEEE European Test Symposium.* pp. 49-54

64 Borri, S., Hage-Hassan, M., Girard, P., Pravossoudovitch, S., Virazel, A., 2003. Defect-Oriented Dynamic Fault Modes for Embedded SRAMs. *In Proc.: 8th IEEE European Test Workshop.* pp. 23-28

65 David, R., Fuentes, A., Courtois, B., 1989. Random Pattern Testing Versus Deterministic Testing of RAM's. *IEEE Transactions on Computers.* 38(5), pp. 637-650

66 Dekker, R., Beenker, F., Thijssen, L, 1988. Fault Modeling and Test Algorithm Development for Static Random Access Memories. *In Proc.: IEEE International Test Conference.* pp. 343-352

67 Dekker, R., Beenker, F., Thijssen, L., 1990. A Realistic Fault Model and Test Algorithms for Random Access Memories. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.* 9(6), pp. 567-572

68 Dilillo, L., Girard, P., Pravossoudovitch, S., Virazel, A., Borri, S., Hage-Hassan, M., 2005. Efficient March Test Procedure for Dynamic Read Destructive Fault Detection in SRAM Memories. *Journal of Electronic Testing: Theory and Applications.* 21(5), pp. 551-561

69 Dilillo, L., Girard, P., Pravossoudovitch, S., Virazel, A., Hage-Hassan, M.B., 2005. Data Retention Faults in SRAM Memories: Analysis and Detection Procedures. *In Proc.: 23rd IEEE VLSI Test Symposium.* pp. 183-188

70 Eder, A., 1998. Embedded Memories in a 32 bit High Performance Microcontroller. *In Proc.: IEEE Symposium on IC/Package Design Integration.* pp. 4-8

71 Franklin, M., Saluja, K.K., 1990. Built-In Self-Testing of Random-Access Memories. *IEEE Computer.* 23(10), pp. 45-56

72 Gupta, A., 2009. *Semiconductor Memory Testing: Fault Models and Test Considerations for High Performance Embedded SRAM's.* VDM Verlag Dr. Müller, Saarbrücken, Germany

73 Hamdioui, S., Gaydadjiev, G.N., van de Goor, A.J., 2004. The State-of-the-art and Future Trends in Testing Embedded Memories. *Records of IEEE Int. Workshop on Memory Technology, Design and Testing.* pp. 54-59

74 Hamdioui, S., Wadsworth, R., Reyes, J.D., van de Goor, A.J., 2004. Memory Faults Modeling Trends: A Case Study. *Journal of Electronic Testing: Theory and Applications.* 20(3), pp. 245-255

75 Hamdioui, S., van de Goor, A.J., 2002. Efficient Tests for Realistic Faults in Dual-Port SRAMs. *IEEE Transactions on Computers.* 51(5), pp. 460-473

76 Hamdioui, S., van de Goor, A.J., Rodgers, M., 2003. Detecting Intra-Word Faults in Word-Oriented Memories. *In Proc.: 21st IEEE VLSI Test Symposium.* pp. 241-247

77 Harutunyan, G., Vardanian, V.A., Zorian, Y., 2006. Minimal March Tests for Dynamic Faults in Random Access Memories. *In Proc.: 11th European Test Symposium.* pp. 43-48

78 Hirabayashi, O., Suzuki, A., Yabe, T., Kawasumi, A., Takeyama, Y., Kushida, K., Tohata, A., Otsuka, N., 2002. DFT Techniques for Wafer-Level At-Speed Testing of High-Speed SRAMs., *In Proc.: IEEE International Test Conference.* pp. 164-169

79 Huang, R.F., Chou, Y.F., Wu, C.W., 2003. Defect Oriented Fault Analysis for SRAM. *In Proc.: 12th IEEE Asian Test Symposium.* pp. 256-261

80 Jee, A., 2002., Defect-Oriented Analysis of Memory BIST Tests. *In Proc.: 8th IEEE On-Line Testing Workshop.* pp. 201-205

81 Jee, A., Colburn, J.E., Irrinki, V.S., Puri, M., 2000. Optimizing Memory Tests by Analyzing Defect Coverage. *Records of IEEE Int. Workshop on Memory Technology, Design and Testing.* pp. 20-25

82 Kim, V.K., Chen, T., 1999. On Comparing Functional Fault Coverage and Defect Coverage for Memory Testing. *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems.* 18(11), pp. 1676-1683

83 Knaizuk, J., Harmann, C.R.P., 1977. An Algorithm for Testing Random Access Memories. *IEEE Transactions on Computers.* C-26(4), pp. 414-416

84 Linder, M., Eder, A., Oberländer, K., Huch, M., Resch, G., 2010. Analysis on Effectiveness of SRAM Test Algorithms and Test Statistics on Industrial Data. *4. GMM/GI/ITG-Fachtagung 'Zuverlässigkeit und Entwurf'*.,13.-.15.9.2010, Wildbad Kreuth, Germany. pp. 49-50

85 Mazumder, P., Patel, J.H., 1989. An Efficient Built-In Self Testing for Random-Access Memory. *IEEE Transactions on Industrial Electronics.* 36(2), pp. 246-453

86 Mikitjuk, V.G., Yarmolik, V.N., van de Goor, A.J., 1996. RAM Testing Algorithms for Detection Multiple Linked Faults. *In Proc.: European Design and Test Conference.* pp. 435-439

87 Mrozek, I., Yarmolik, V.N., 2008. MATS+ transparent memory test for pattern sensitive fault detection. *IEEE 15th Conference on Mixed Design of Integrated Circuits and Systems.* pp. 493-498

88 Mrozek, I., Yarmolik, V.N., 2008. Optimal Backgrounds Selection for Multi Run Memory Testing. *Computer Information Systems and Industrial Management Applications.* pp. 155-156

89 Naik, S., Agricola, F., Maly, W., 1993. Fault Analysis of High-Density CMOS SRAMs. *IEEE Desing & Test of Computers.* 10(2), pp. 13-23

90 Nair, R., Thatte, S.M., Abraham, J.A., 1978. Efficient Algorithms for Testing Semiconductor Random-Access Memories. *IEEE Transactions on Computers.* C-27(6), pp. 572-576

91 Rajsuman, R., 1991. New Algorithm for Testing Random Access Memories. *Electronics Letters*. 27(7), pp. 574-575

92 Riedel, M., Rajski., 1995. Fault Coverage Analysis of RAM Test Algorithms. *In Proc.: 13th IEEE VLSI Test Symposium*. pp. 227-234

93 Sarma, D., Papachristou, C.A., Saifuddin, F.T., 1982. Fault coverage of pattern-sensitive fault-detection algorithms for semiconductor memories. *Electronics Letters*. 18(22), pp. 950-951

94 Tsai, P.C., Wang S.J., Chang, F.M., 2005. FSM-Based Programmable Memory BIST with Macro Command. *IEEE Int. Workshop on Memory Technology, Design and Testing*. pp. 72-77

95 Tseng, C.W., Mitra, S., Davidson, S., McCluskey, E.J., 2001. An Evaluation of Pseudo Ranodm Testing for Detecting Real Defects. *In Proc.: 19th IEEE VLSI Test Symposium*. pp. 404-409

96 van de Goor, A.J., 2004. An Industrial Evaluation of DRAM Tests. *IEEE Design & Test of Computers*. 21(5), pp. 430-440

97 van de Goor, A.J., Gaydadjiev, G.N., 1997. An Analysis of (Linked) Address Decoder Faults. *In Proc.: IEEE Int. Workshop on Memory Technology, Design and Testing*. pp. 13-20

98 van de Goor, A.J., Hamdioui, S., Al-Ars, Z., 2004. The Effectiveness of Scan Test and its new Variants. *Records of IEEE Int. Workshop on Memory Technology, Design and Testing*. pp. 26-31

99 van de Goor, A.J., Lin, M., 1997. The Implementation of Pseudo-Random Memory Tests on Commercial Memory Testers. *In Proc.: IEEE International Test Conference*. pp. 226-235

100 van de Goor, A.J., Tlili, I.B.S., 1997. Disturb Neighborhood Pattern Sensitivity Fault. *In Proc.: 15th IEEE VLSI Test Symposium*. pp. 37-45

101 van de Goor, A.J., Tlili, I.B.S., 1998. March Tests for Word-Oriented Memories. *In Proc.: Design and Test in Europe*. pp. 501-508

102 van de Goor, A.J., Tlili, I.B.S., 2003. A Systematic Method for Modifying March Tests for Bit-Oriented Memories into Tests for Word-Oriented Memories. *IEEE Transactions on Computers*. 52(10), pp. 1320-1331

103 van de Goor, A.J., Tlili, I.B.S., Hamdioui, S., 1998. Converting March Tests for Bit-Oriented Memories into Test for Word-Oriented Memories. *Records of IEEE Int. Workshop on Memory Technology, Design and Testing*. pp. 46-52

104 Veenstra, P.K., Beenker, F.P.M., Koomen, J.J.M., 1988. Testing of random access memories: theory and practice. *Electronic Circuits & Systems*. 135(1), pp. 24-28

105 Wang, B., Yang, J., Ivanov, A., 2003. Reducing Test Time of Embedded SRAMs. *Records of IEEE Int. Workshop on Memory Technology, Design and Testing*. pp. 52-57

106 Wang, C.W., Wu, C.F., Li, J.F., Wu, C.W., Teng, T., Chiu, K., Lin, H.P., 2000. A Built-In Self-Test and Self-Diagnosis Scheme for Embedded SRAM. *In Proc.: 9th IEEE Asian Test Symposium*. pp. 45-50

107 Wu, C.F., Huang, C.T., Cheng, K.L., Wu, C.W., 2000. Simulation-Based Test Algorithm Generation for Random Access Memories. *In Proc.: 18th IEEE VLSI Test Symposium*. pp. 291-296

108 Zarrineh, K., Deo, A.P., Adams, R.D., 2000. Defect Analysis and Realistic Fault Model Extensions for Static Random Access Memories. *Records of IEEE Int. Workshop on Memory Technology, Design and Testing.* pp. 119-124

109 Zarrineh, K., Upadhyaya, S.J., 1999. On Programmable Memory Built-In Self Test Architectures. *In Proc.: Design, Automation and Test in Europe.* pp. 708-713

Appendix A
Additional Results of Chapter 6

This chapter contains additional tables and figures that show the test results for the fault coverage of test algorithms for each test, and the tables that summarize the test results for efficiency of pairs of algorithms.

A.1 Fault Coverage of Algorithms

This section contains the test results of all tests for all of the seven test number in the study. The total fault coverage and number of exclusive faults for each algorithm in the study test set is listed and additionally, the fault coverage is shown graphically. 100% always refers to the total number of faults detected at the respective test number.

The results of TN1522 and TN1622 are of special interest as these tests are performed before Burn-In. Hence, the expected fault coverage of RESET was zero. Nevertheless, RESET detects 33 faults at TN1522 and 11 faults at TN 1622 although the RESET configuration already ran at wafer test. It has to be assumed that these faults are caused by the packaging process or due to handling between wafer and Burn-In test.

TN1522 (1.35V / +125°C)

Table A.1. Fault coverage at TN1522

Algorithm	|F|	FC	excl.	Algorithm	|F|	FC	excl.
Total	***617***			March Y	130	21%	0
SCAN	50	8%	0	March LR	286	46%	0
SCAN+	59	10%	0	March LA	319	52%	0
MATS	83	13%	0	March RAW	447	72%	1
MATS+	112	18%	0	March RAW1	101	16%	0
MATS++	108	18%	0	March AB	297	48%	0
March C-	116	19%	0	March AB1	341	55%	3
March A	120	19%	0	March BDN	318	52%	0
March B	130	21%	0	March SR	276	45%	0
Algorithm B	289	47%	0	March SS	134	22%	0
March C+	124	20%	0	BLIF	204	33%	0
PMOVI	127	21%	0	Ham5R	346	56%	5
March 1/0	120	19%	0	Ham5W	298	48%	2
March TP	121	20%	0	March G	163	29%	0
March U	280	45%	0	Ham_Walk	384	62%	2
March X	110	18%	0	RESET	33	5%	0

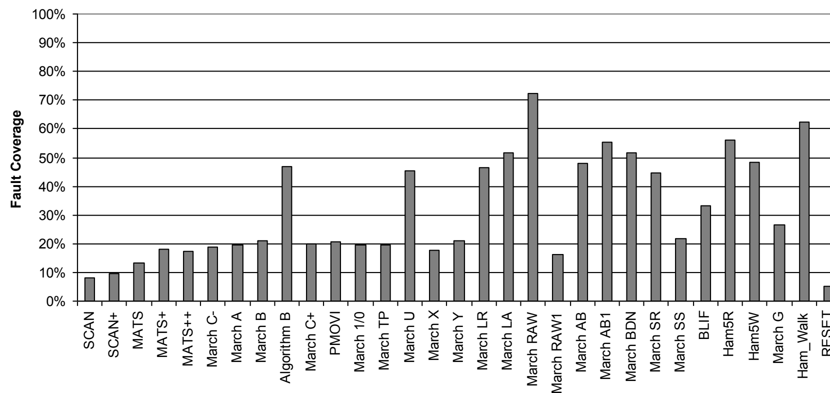

Figure A.1. Fault coverage at TN1522

TN1622 (1.80V / +125°C)

Table A.2. Fault coverage at TN1622

Algorithm	|F|	FC	excl.	Algorithm	|F|	FC	excl.
Total	***56***			March Y	24	43%	0
SCAN	18	32%	0	March LR	30	54%	0
SCAN+	17	30%	0	March LA	29	52%	0
MATS	21	38%	0	March RAW	46	82%	0
MATS+	21	38%	0	March RAW1	22	39%	0
MATS++	24	43%	0	March AB	27	48%	0
March C-	23	41%	0	March AB1	24	43%	1
March A	25	45%	0	March BDN	27	48%	0
March B	26	46%	0	March SR	29	52%	0
Algorithm B	32	57%	0	March SS	26	46%	0
March C+	26	46%	0	BLIF	12	21%	0
PMOVI	27	48%	0	Ham5R	25	45%	0
March 1/0	24	43%	0	Ham5W	23	41%	0
March TP	28	50%	0	March G	28	50%	0
March U	31	55%	0	Ham_Walk	33	59%	0
March X	23	41%	0	RESET	11	20%	0

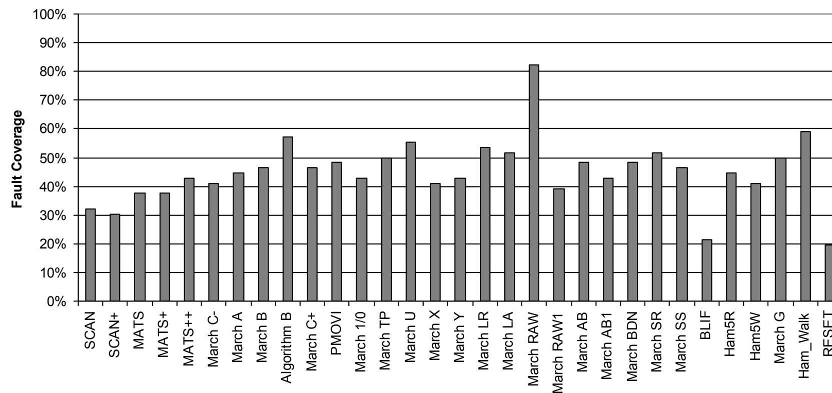

Figure A.2. Fault coverage at TN1622

TN6531 (1.35V / +145°C)

Table A.3. Fault coverage at TN6531

Algorithm	|F|	FC	excl.	Algorithm	|F|	FC	excl.
Total	***2439***			March Y	1056	43%	0
SCAN	684	28%	0	March LR	1921	49%	1
SCAN+	727	30%	0	March LA	1414	58%	1
MATS	937	38%	0	March RAW	1563	64%	4
MATS+	1047	43%	0	March RAW1	1020	42%	0
MATS++	1037	43%	0	March AB	1402	57%	0
March C-	1092	45%	0	March AB1	792	32%	5
March A	1109	45%	0	March BDN	1429	59%	0
March B	1148	47%	1	March SR	1898	78%	0
Algorithm B	1908	78%	0	March SS	1115	46%	0
March C+	1074	44%	0	BLIF	1047	43%	0
PMOVI	1093	45%	0	Ham5R	783	32%	10
March 1/0	1083	44%	0	Ham5W	473	19%	0
March TP	1110	46%	0	March G	1232	51%	4
March U	1909	78%	1	Ham_Walk	2063	85%	22
March X	1067	44%	0	RESET	540	22%	0

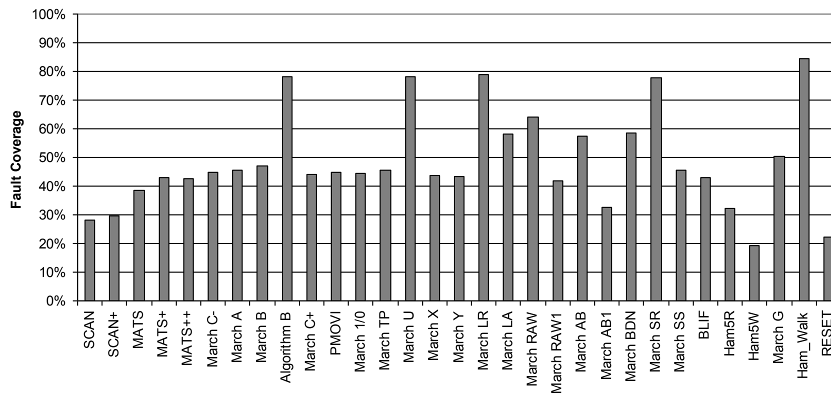

Figure A.3. Fault coverage at TN6531

TN6631 (1.80V / +145°C)

Table A.4. Fault coverage at TN6631

Algorithm	|F|	FC	excl.	Algorithm	|F|	FC	excl.
Total	***175***			March Y	116	66%	0
SCAN	83	47%	0	March LR	128	73%	0
SCAN+	94	54%	0	March LA	126	72%	0
MATS	99	57%	0	March RAW	153	87%	0
MATS+	103	59%	0	March RAW1	113	65%	0
MATS++	99	57%	0	March AB	121	69%	0
March C-	114	65%	0	March AB1	72	41%	0
March A	109	62%	0	March BDN	120	69%	0
March B	110	63%	0	March SR	122	70%	0
Algorithm B	123	70%	0	March SS	115	66%	0
March C+	123	70%	0	BLIF	29	17%	0
PMOVI	120	49%	0	Ham5R	82	47%	0
March 1/0	114	45%	0	Ham5W	41	23%	0
March TP	116	44%	0	March G	120	69%	2
March U	123	70%	0	Ham_Walk	128	73%	0
March X	104	59%	0	RESET	70	40%	0

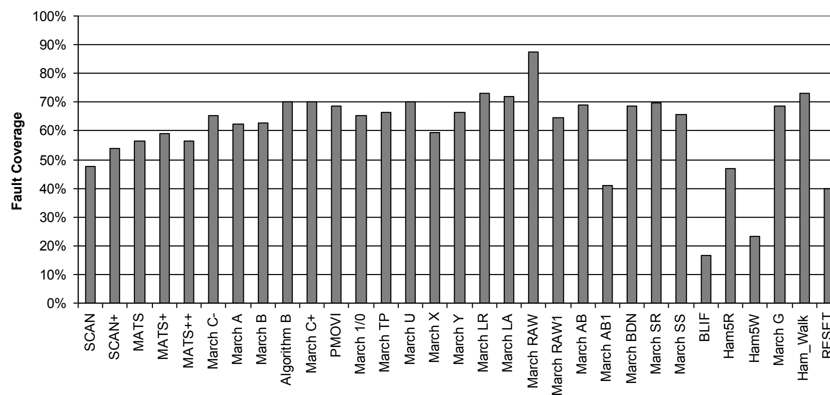

Figure A.4. Fault coverage at TN6631

TN3741 (1.30V / -40°C)

Table A.5. Fault coverage at TN3741

Algorithm	|F|	FC	excl.	Algorithm	|F|	FC	excl.
Total	***237***			March Y	124	52%	0
SCAN	79	33%	0	March LR	113	48%	0
SCAN+	98	41%	0	March LA	165	70%	0
MATS	82	35%	0	March RAW	199	84%	0
MATS+	105	44%	0	March RAW1	130	55%	1
MATS++	107	45%	0	March AB	184	78%	0
March C-	109	46%	0	March AB1	166	70%	3
March A	111	47%	0	March BDN	186	78%	2
March B	125	53%	0	March SR	113	48%	1
Algorithm B	124	52%	1	March SS	145	61%	0
March C+	134	57%	0	BLIF	68	29%	0
PMOVI	131	55%	0	Ham5R	163	69%	2
March 1/0	127	54%	0	Ham5W	113	48%	0
March TP	130	55%	1	March G	159	67%	1
March U	125	53%	0	Ham_Walk	154	65%	1
March X	103	43%	0	RESET	99	42%	0

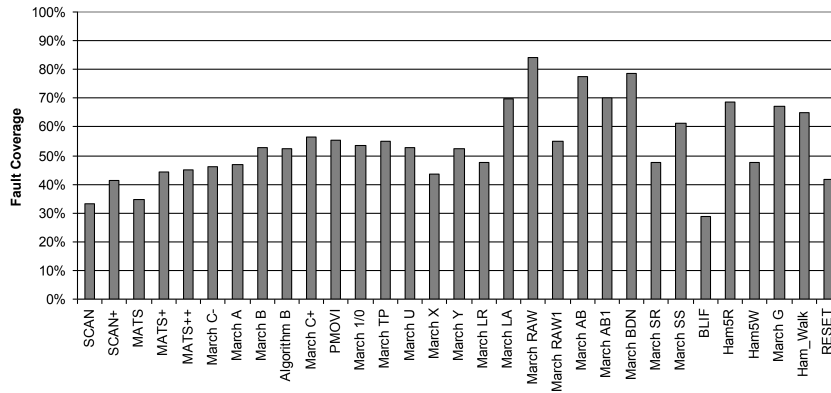

Figure A.5. Fault coverage at TN3741

TN3841 (1.50V / -40°C)

Table A.6. Fault coverage at TN3841

Algorithm	|F|	FC	excl.	Algorithm	|F|	FC	excl.
Total	***70***			March Y	17	24%	0
SCAN	10	14%	0	March LR	13	19%	0
SCAN+	13	19%	0	March LA	56	80%	0
MATS	10	14%	0	March RAW	62	89%	0
MATS+	13	19%	0	March RAW1	14	20%	0
MATS++	16	23%	0	March AB	56	80%	0
March C-	13	19%	0	March AB1	60	86%	2
March A	11	16%	0	March BDN	53	76%	0
March B	15	21%	0	March SR	15	21%	0
Algorithm B	14	20%	0	March SS	19	27%	0
March C+	14	20%	0	BLIF	9	13%	0
PMOVI	16	23%	0	Ham5R	58	83%	0
March 1/0	18	26%	0	Ham5W	50	71%	0
March TP	13	19%	0	March G	17	24%	0
March U	14	20%	0	Ham_Walk	41	59%	0
March X	12	17%	0	RESET	10	14%	0

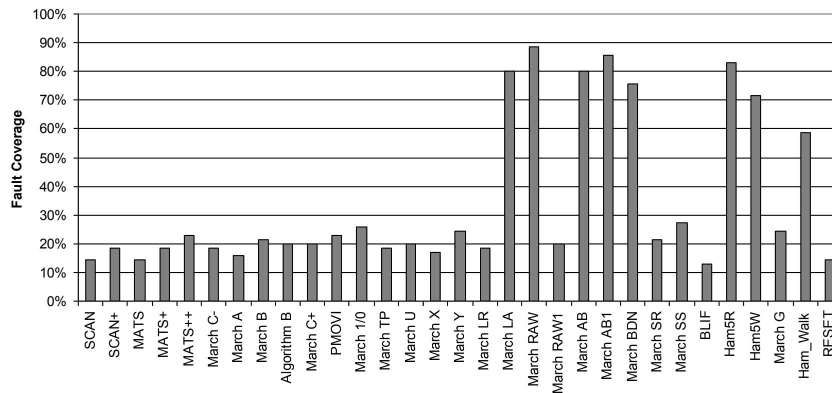

Figure A.6. Fault coverage at TN3841

TN3941 (1.80V / -40°C)

Table A.7. Fault coverage at TN3941

Algorithm	|F|	FC	excl.	Algorithm	|F|	FC	excl.
Total	***46***			March Y	19	41%	0
SCAN	21	46%	0	March LR	21	46%	0
SCAN+	20	43%	0	March LA	36	78%	0
MATS	20	43%	0	March RAW	37	80%	0
MATS+	20	43%	0	March RAW1	21	46%	0
MATS++	19	41%	0	March AB	37	80%	0
March C-	20	43%	0	March AB1	37	80%	0
March A	20	43%	0	March BDN	35	76%	0
March B	22	48%	0	March SR	22	48%	0
Algorithm B	20	43%	0	March SS	22	48%	0
March C+	22	48%	0	BLIF	19	41%	0
PMOVI	22	48%	0	Ham5R	40	87%	0
March 1/0	21	46%	0	Ham5W	32	70%	0
March TP	20	43%	0	March G	23	50%	0
March U	22	48%	0	Ham_Walk	28	61%	0
March X	20	43%	0	RESET	20	43%	0

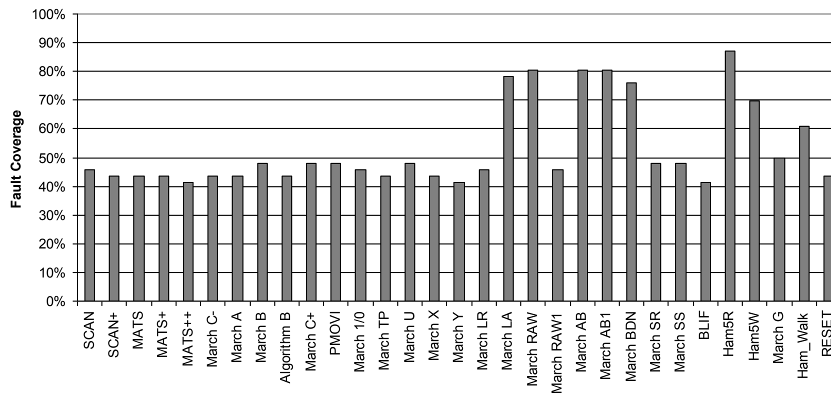

Figure A.7. Fault coverage at TN3941

TN4441 (1.30V / +25°C)

Table A.8. Fault coverage at TN4441

Algorithm	|F|	FC	excl.	Algorithm	|F|	FC	excl.
Total	***165***			March Y	51	31%	0
SCAN	34	21%	0	March LR	55	33%	0
SCAN+	36	22%	0	March LA	118	72%	0
MATS	31	19%	0	March RAW	139	84%	1
MATS+	44	27%	0	March RAW1	54	33%	0
MATS++	50	30%	0	March AB	123	75%	0
March C-	51	31%	0	March AB1	118	72%	0
March A	49	30%	0	March BDN	120	73%	0
March B	51	31%	0	March SR	53	32%	0
Algorithm B	56	34%	0	March SS	63	38%	0
March C+	56	34%	0	BLIF	24	15%	0
PMOVI	57	35%	0	Ham5R	130	79%	0
March 1/0	58	35%	0	Ham5W	98	59%	0
March TP	49	30%	0	March G	63	38%	0
March U	61	37%	0	Ham_Walk	95	58%	2
March X	44	27%	0	RESET	40	24%	0

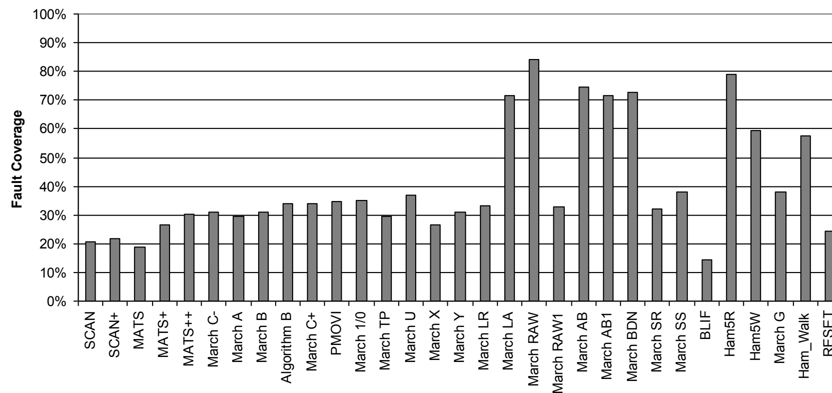

Figure A.8. Fault coverage at TN4441

TN4541 (1.80V / +25°C)

Table A.9. Fault coverage at TN4541

Algorithm	|F|	FC	excl.	Algorithm	|F|	FC	excl.
Total	***25***			March Y	5	20%	0
SCAN	7	28%	0	March LR	8	32%	0
SCAN+	7	28%	0	March LA	19	76%	0
MATS	4	16%	0	March RAW	21	84%	0
MATS+	4	16%	0	March RAW1	5	20%	0
MATS++	6	24%	0	March AB	17	68%	0
March C-	6	24%	0	March AB1	17	68%	0
March A	7	28%	0	March BDN	14	56%	0
March B	7	28%	0	March SR	8	32%	0
Algorithm B	7	28%	0	March SS	8	32%	0
March C+	8	32%	0	BLIF	3	12%	0
PMOVI	6	24%	0	Ham5R	18	72%	0
March 1/0	7	28%	0	Ham5W	18	72%	0
March TP	7	28%	0	March G	8	32%	0
March U	9	26%	0	Ham_Walk	10	40%	0
March X	5	20%	0	RESET	6	24%	0

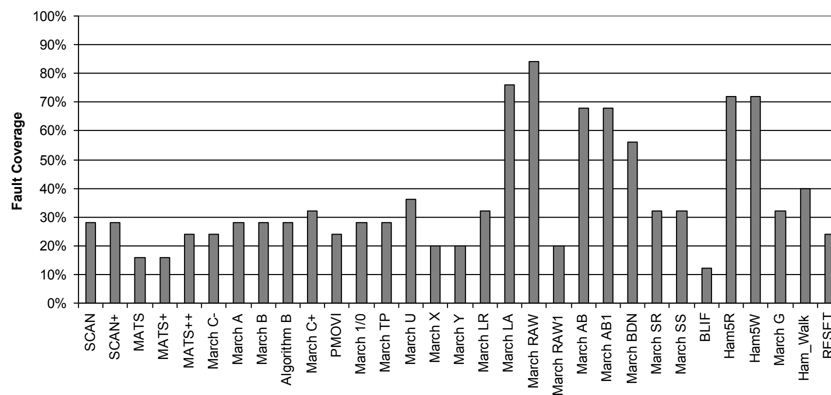

Figure A.9. Fault coverage at TN4541

A.2 Effectiveness of Pairs of Algorithms

In this section, the test results of the analysis for effectiveness of pairs of algorithms are given for all seven tests during the study. For each test number, a table is derived that contains fault coverage, union and intersection. The colors represent Q_{Eff} as defined in section Chapter 6 (see Fig. 6.3).

TN6531 (uncolored, values of QEff)

Table A.10. Values of Q_{Eff} at TN6531

	SCN	SCNP	MTS	MTSP	MTSPF	CM	A	B	AlgB	CP	PMOVI	1/0	TP	U	X	Y	LR	LA	RAW	RAW1	AB	AB1	BDN	SR	SS	BLIF	Ham5R	Ham5W	G	HamWk
SCN	**684**	0,937	0,919	0,941	0,934	0,988	0,991	0,995	0,990	0,969	0,951	0,955	0,974	0,994	0,946	0,939	0,992	0,987	0,975	0,950	0,970	0,246	0,989	0,992	0,988	0,018	0,244	0,016	1,000	0,992
SCNP		**727**	0,886	0,906	0,901	0,975	0,982	0,982	0,979	0,939	0,935	0,929	0,966	0,979	0,917	0,895	0,983	0,971	0,948	0,929	0,965	0,223	0,967	0,985	0,955	0,015	0,222	0,016	1,000	0,981
MTS			**937**	0,900	0,909	0,929	0,941	0,958	0,983	0,894	0,902	0,899	0,941	0,979	0,890	0,877	0,967	0,931	0,919	0,852	0,917	0,229	0,943	0,964	0,908	0,134	0,226	0,037	0,983	0,986
MTSP				**1047**	0,901	0,902	0,916	0,928	0,971	0,878	0,867	0,886	0,899	0,978	0,898	0,867	0,972	0,923	0,918	0,845	0,906	0,323	0,931	0,959	0,891	0,169	0,321	0,133	0,964	0,980
MTSPP					**1037**	0,914	0,937	0,938	0,978	0,891	0,880	0,901	0,916	0,975	0,892	0,890	0,969	0,941	0,936	0,861	0,924	0,312	0,940	0,962	0,911	0,168	0,311	0,131	0,974	0,984
CM						**1092**	0,897	0,918	0,970	0,897	0,868	0,887	0,892	0,969	0,883	0,868	0,963	0,925	0,907	0,899	0,909	0,334	0,926	0,953	0,898	0,175	0,342	0,158	0,974	0,974
A							**1109**	0,931	0,970	0,897	0,871	0,886	0,891	0,963	0,902	0,901	0,962	0,913	0,892	0,926	0,894	0,333	0,920	0,959	0,866	0,178	0,327	0,138	0,975	0,968
B								**1148**	0,956	0,922	0,897	0,909	0,902	0,949	0,928	0,923	0,951	0,882	0,873	0,946	0,865	0,343	0,895	0,947	0,890	0,197	0,342	0,148	0,949	0,956
AlgB									**1908**	0,987	0,962	0,977	0,959	0,925	0,970	0,974	0,933	0,750	0,657	0,978	0,756	0,534	0,743	0,919	0,961	0,936	0,509	0,326	0,921	0,960
CP										**1074**	0,897	0,905	0,904	0,988	0,861	0,890	0,981	0,946	0,939	0,900	0,939	0,340	0,963	0,980	0,929	0,188	0,346	0,162	0,971	0,990
PMOVI											**1093**	0,891	0,879	0,962	0,852	0,873	0,966	0,916	0,907	0,894	0,902	0,342	0,930	0,962	0,890	0,193	0,348	0,164	0,953	0,970
1/0												**1083**	0,893	0,974	0,857	0,895	0,974	0,938	0,938	0,888	0,933	0,342	0,945	0,971	0,913	0,190	0,346	0,164	0,962	0,979
TP													**1110**	0,959	0,877	0,877	0,961	0,906	0,902	0,903	0,889	0,329	0,921	0,951	0,864	0,186	0,334	0,152	0,960	0,968
U														**1909**	0,977	0,972	0,932	0,750	0,657	0,982	0,756	0,538	0,753	0,925	0,965	0,943	0,510	0,330	0,919	0,966
X															**1067**	0,863	0,958	0,901	0,907	0,883	0,910	0,337	0,929	0,951	0,886	0,173	0,333	0,137	0,961	0,973
Y																**1056**	0,970	0,922	0,931	0,875	0,914	0,335	0,944	0,959	0,900	0,179	0,331	0,145	0,962	0,973
LR																	**1921**	0,748	0,659	0,979	0,753	0,536	0,751	0,928	0,972	0,958	0,512	0,332	0,917	0,960
LA																		**1414**	0,910	0,939	0,908	0,707	0,920	0,752	0,929	0,276	0,677	0,622	0,820	0,815
RAW																			**1563**	0,935	0,908	0,910	0,899	0,657	0,936	0,321	0,904	0,935	0,820	0,719
RAW1																				**1020**	0,945	0,320	0,962	0,975	0,929	0,163	0,314	0,125	0,977	0,988
AB																					**1402**	0,710	0,926	0,757	0,918	0,280	0,670	0,603	0,820	0,826
AB1																						**792**	0,718	0,544	0,357	0,198	0,691	0,794	0,369	0,616
BDN																							**1429**	0,748	0,941	0,284	0,682	0,623	0,830	0,816
SR																								**1898**	0,958	0,931	0,514	0,338	0,916	0,954
SS																									**1115**	0,205	0,367	0,184	0,943	0,974
BLIF																										**1047**	0,181	0,295	0,231	0,971
Ham5R																											**783**	0,950	0,369	0,562
Ham5W																												**473**	0,176	0,398
G																													**1232**	0,943
HamWk																														**2063**

TN1522 (1.35V / +125°C)

Table A.11. Union and intersection at TN1522

	SCN	SCNP	MTS	MTSP	MTSPP	CM	A	B	AlgB	CP	PMOVI	1/0	TP	U	X	Y	LR	LA	RAW	RAW1	AB	AB1	BDN	SR	SS	BLIF	Ham5R	Ham5W	G	HamWk
HamWk	48	54	82	108	105	111	112	123	274	115	121	117	115	270	106	122	273	234	251	98	220	163	235	262	124	198	144	116	149	**384**
G	50	57	82	108	101	113	116	126	156	115	120	114	110	147	102	123	151	126	134	99	119	49	129	153	123	86	48	26	**163**	398
Ham5W	4	4	9	17	16	20	18	20	52	24	27	26	22	54	20	22	49	179	285	17	173	266	177	47	26	49	287	**298**	435	566
Ham5R	13	13	18	33	31	35	35	35	80	39	46	44	34	80	39	40	78	202	319	31	191	299	201	75	44	67	**346**	357	461	586
BLIF	4	7	36	58	53	58	55	62	194	59	69	66	58	187	57	70	195	84	103	46	81	67	92	188	68	**204**	483	453	281	390
SS	47	50	75	97	94	100	103	110	125	106	108	106	102	126	94	108	127	112	121	89	109	41	115	127	**134**	270	436	406	174	394
SR	49	56	82	110	103	115	117	126	262	120	124	117	118	258	109	127	261	145	161	99	137	74	153	**276**	283	292	547	527	286	398
BDN	49	52	78	103	96	105	107	113	154	108	113	112	106	155	101	118	153	287	300	94	283	217	**318**	441	337	430	463	439	352	467
AB1	11	12	16	34	29	37	34	36	79	36	45	43	35	80	37	39	77	217	321	30	206	**341**	442	543	434	478	388	373	455	562
AB	47	50	77	96	92	98	102	106	139	106	109	105	104	141	90	107	139	279	285	86	**297**	432	332	436	322	420	452	422	341	461
RAW1	46	50	67	82	82	88	90	92	100	88	89	87	86	100	85	91	97	91	92	**101**	312	412	325	278	146	259	416	382	165	387
RAW	49	53	78	103	100	106	108	117	168	114	118	113	111	168	97	120	164	296	**447**	456	459	467	465	562	460	548	474	460	476	580
LA	49	55	79	98	97	103	106	110	149	107	116	110	107	149	98	110	145	**319**	470	329	337	443	350	450	341	439	463	438	356	469
LR	50	53	82	112	103	114	114	124	269	118	123	118	115	260	107	126	**286**	460	569	290	444	550	451	301	293	295	554	535	298	397
Y	47	49	74	104	95	99	103	107	126	106	109	106	100	124	98	**130**	290	339	457	140	320	432	330	279	156	264	436	406	170	392
X	44	50	70	93	88	95	96	95	107	90	94	88	94	108	**110**	142	289	331	460	126	317	414	327	277	150	257	417	388	171	388
U	48	55	82	109	105	114	113	123	263	119	121	117	117	**280**	282	286	306	450	559	281	436	541	443	298	288	297	546	524	296	394
TP	46	52	78	92	91	101	101	104	118	103	99	97	**121**	284	137	151	292	333	457	136	314	427	333	279	153	267	433	397	174	390
1/0	48	47	73	95	89	98	94	99	116	102	107	**120**	144	283	142	144	288	329	454	134	312	418	326	279	148	258	422	392	169	387
PMOVI	45	51	76	96	92	100	102	105	122	105	**127**	140	149	286	143	148	290	330	456	139	315	423	332	279	153	262	427	398	170	390
CP	48	52	74	93	89	99	96	102	117	**124**	146	142	142	285	144	148	292	336	457	137	315	429	334	280	152	269	431	398	172	393
AlgB	48	55	82	111	104	113	119	127	**289**	296	294	293	292	306	292	293	306	459	568	290	447	551	453	303	298	299	555	535	296	399
B	47	53	77	97	95	101	112	**130**	292	152	152	151	147	287	145	153	292	339	460	139	321	435	335	280	154	272	441	408	167	391
A	47	54	76	96	94	101	**120**	138	290	148	145	146	140	287	134	147	292	333	459	131	315	427	331	279	151	269	431	400	167	392
CM	47	52	78	95	88	**116**	135	145	292	141	143	138	136	282	131	147	288	332	457	129	315	420	329	277	150	262	427	394	166	389
MTSPF	46	46	73	90	**108**	136	134	143	293	143	143	139	138	283	130	143	291	330	455	127	313	420	330	281	148	259	423	390	170	387
MTSP	46	48	78	**112**	130	133	136	145	290	143	143	137	141	283	129	138	286	333	456	131	313	419	327	278	149	258	425	393	167	388
MTS	43	46	**83**	117	118	121	127	136	290	133	134	130	126	281	123	139	287	323	452	117	303	408	323	277	142	251	411	372	164	385
SCNP	44	**59**	96	123	121	123	125	136	293	131	135	132	128	284	119	140	292	323	453	110	306	388	325	279	143	256	392	353	165	389
SCN	**50**	65	90	116	112	119	123	133	291	126	132	122	125	282	116	133	286	320	448	105	300	380	319	277	137	250	383	344	163	386

TN1622 (1.80V / +125°C)

Table A.12. Union and intersection at TN1622

	SCN	SCNP	MTS	MTSP	MTSPP	CM	A	B	AlgB	CP	PMOVI	1/0	TP	U	X	Y	LR	LA	RAW	RAW1	AB	AB1	BDN	SR	SS	BLIF	Ham5R	Ham5W	G	HamWk
HamWk	18	17	20	20	23	23	25	26	31	25	26	23	27	31	23	23	30	27	26	21	25	8	25	29	25	12	8	6	27	33
G	18	17	21	21	24	23	25	26	26	25	26	23	27	27	22	23	26	27	26	22	26	8	26	25	26	7	8	6	28	34
Ham5W	4	4	5	6	5	6	6	6	7	6	6	6	6	6	6	6	6	7	21	6	7	17	7	6	6	6	20	23	45	50
Ham5R	6	6	7	8	7	8	8	8	9	8	8	8	8	8	8	8	8	9	25	8	9	17	9	8	8	6	25	28	45	50
BLIF	4	4	6	7	6	7	7	7	12	8	8	8	8	11	8	8	11	7	8	7	7	5	7	11	7	12	31	29	33	33
SS	18	16	20	20	23	22	23	24	24	24	25	22	25	25	20	22	24	25	25	21	25	8	25	23	26	31	43	43	28	34
SR	17	17	20	20	22	23	24	25	29	24	24	23	25	29	23	22	28	24	25	21	24	8	24	29	32	30	46	46	32	33
BDN	17	16	21	21	24	23	23	25	25	24	25	23	25	25	21	23	24	26	27	21	27	9	27	32	28	32	43	43	29	35
AB1	7	6	6	7	7	8	8	8	9	8	8	8	8	8	7	7	7	8	21	8	9	24	42	45	42	31	32	30	44	49
AB	17	16	21	21	24	23	23	25	25	24	25	23	25	25	21	23	24	26	27	21	27	42	27	32	28	32	43	43	29	35
RAW1	16	16	19	19	20	20	20	21	21	21	21	20	22	21	19	19	20	21	21	22	28	38	28	30	27	27	39	39	28	34
RAW	17	16	21	21	24	23	23	25	27	24	26	23	26	26	21	24	25	26	46	47	46	49	46	50	47	50	46	48	48	53
LA	17	17	21	21	23	22	24	25	25	24	25	22	26	26	22	23	26	29	49	30	30	45	30	34	30	34	45	45	30	35
LR	17	17	20	20	22	22	24	25	29	24	25	22	26	30	23	23	30	33	51	32	33	47	33	31	32	31	47	47	32	33
Y	15	15	20	21	22	21	21	22	23	22	24	22	24	23	21	24	31	30	46	27	28	41	28	31	28	28	41	41	29	34
X	15	16	19	20	20	21	22	22	23	21	21	21	22	23	23	26	30	30	48	26	29	40	29	29	29	27	40	40	29	33
U	18	17	20	20	23	23	25	26	30	25	26	23	27	31	31	32	31	34	51	32	33	47	33	31	32	32	48	48	32	33
TP	18	17	21	21	24	23	24	25	26	25	27	23	28	32	29	28	32	31	48	28	30	44	30	32	29	32	45	45	29	34
1/0	16	15	20	21	22	22	22	22	23	24	23	24	29	32	26	26	32	31	47	26	28	40	28	30	28	28	41	41	29	34
PMOVI	18	16	21	21	24	23	23	24	25	25	27	28	28	32	29	27	32	31	47	28	29	43	29	32	28	31	44	44	29	34
CP	18	16	21	21	23	23	23	23	24	26	28	26	29	32	28	28	32	31	48	27	29	42	29	31	28	30	43	43	29	34
AlgB	17	17	20	20	23	23	24	26	32	34	34	33	34	33	32	33	33	36	51	33	34	47	34	32	34	32	48	48	34	34
B	17	17	20	20	23	23	24	26	32	29	29	28	29	31	27	28	31	30	47	27	28	42	28	30	28	31	43	43	28	33
A	17	16	19	20	21	22	25	27	33	28	29	27	29	31	26	28	31	30	48	27	29	41	29	30	28	30	42	42	28	33
CM	17	16	20	20	22	23	26	26	32	26	27	25	28	31	25	26	31	30	46	25	27	39	27	29	27	28	40	40	28	33
MTSPF	17	16	21	20	24	25	28	27	33	27	27	26	28	32	27	26	32	30	46	26	27	41	27	31	27	30	42	42	28	34
MTSP	14	14	19	21	25	24	26	27	33	26	27	24	28	32	24	24	31	29	46	24	27	38	27	30	27	26	38	38	28	34
MTS	16	16	21	23	24	24	27	27	33	26	27	25	28	32	25	25	31	29	46	24	27	39	27	30	27	27	39	39	28	34
SCNP	15	17	22	24	25	24	26	26	32	27	28	26	28	31	24	26	30	29	47	23	28	35	28	29	27	25	36	36	28	33
SCN	18	20	23	25	25	24	26	27	33	26	27	26	28	31	26	27	31	30	47	24	28	35	28	30	26	26	37	37	28	33

TN6531 (1.35V / +145°C)

Table A.13. Union and intersection at TN6531

	SCN	SCNP	MTS	MTSP	MTSPF	CM	A	B	AlgB	CP	PMOVI	1/0	TP	U	X	Y	LR	LA	RAW	RAW1	AB	AB1	BDN	SR	SS	BLIF	Ham5R	Ham5W	G	HamWk
HamWk	680	717	928	1033	1026	1073	1086	1115	1868	1067	1071	1068	1087	1875	1048	1037	1880	1246	1279	1012	1246	546	1261	1851	1096	1027	500	212	1187	2063
G	684	727	928	1026	1022	1077	1094	1117	1171	1057	1065	1061	1086	1169	1044	1034	1168	1103	1097	1007	1103	388	1110	1167	1081	374	383	108	1232	2108
Ham5W	13	12	26	86	85	100	88	93	178	103	104	104	96	180	88	93	181	325	449	82	317	407	325	184	115	179	458	473	1597	2324
Ham5R	258	258	273	354	346	369	355	365	465	374	374	373	361	466	363	362	467	599	731	350	595	639	602	469	389	218	783	798	1632	2346
BLIF	20	18	212	303	298	307	310	335	1003	328	334	331	322	1008	307	317	1018	417	462	283	424	240	427	999	349	1047	1612	1341	1905	2083
SS	679	707	888	985	987	1033	1029	1049	1087	1034	1029	1033	1029	1090	1001	999	1095	1069	1072	981	1062	386	1077	1085	1115	1813	1509	1473	1266	2082
SR	680	719	914	1019	1011	1059	1080	1109	1818	1060	1066	1063	1075	1824	1033	1028	1827	1189	1215	1003	1183	498	1199	1898	1928	1946	2212	2187	1963	2110
BDN	679	711	904	1004	1000	1045	1057	1078	1193	1051	1048	1048	1059	1203	1022	1021	1201	1355	1350	997	1348	632	1429	2128	1467	2049	1610	1577	1551	2231
AB1	259	258	280	361	352	367	365	371	490	374	374	375	362	493	372	371	491	626	743	361	628	792	1589	2192	1521	1599	936	858	1636	2309
AB	670	710	889	988	990	1034	1040	1057	1182	1036	1030	1041	1038	1182	1010	1002	1179	1334	1331	987	1402	1566	1483	2117	1455	2025	1590	1558	1531	2219
RAW1	663	696	859	933	943	964	979	990	1005	965	961	958	966	1008	955	951	1006	983	979	1020	1435	1451	1452	1915	1154	1784	1453	1411	1245	2071
RAW	672	701	888	994	996	1030	1036	1059	1215	1034	1031	1042	1044	1214	1006	1011	1216	1344	1563	1604	1634	1612	1642	2246	1606	2148	1615	1587	1698	2347
LA	678	713	897	999	1001	1044	1053	1069	1187	1040	1039	1044	1049	1187	1004	1007	1184	1414	1633	1451	1482	1580	1488	2123	1460	2044	1598	1562	1543	2231
LR	680	718	916	1028	1016	1066	1082	1112	1842	1061	1069	1065	1082	1842	1038	1035	1921	2151	2268	1935	2144	2222	2149	1992	1941	1950	2237	2213	1985	2104
Y	658	680	872	972	976	980	1000	1012	1038	994	983	997	985	1037	978	1056	1942	1463	1608	1125	1456	1477	1464	1926	1172	1786	1477	1436	1254	2082
X	661	690	879	990	977	1000	1011	1026	1046	987	981	984	996	1051	1067	1145	1950	1477	1624	1132	1459	1487	1474	1932	1181	1807	1487	1452	1255	2082
U	681	716	924	1032	1020	1070	1083	1111	1834	1066	1066	1065	1081	1909	1925	1928	1988	2136	2258	1921	2129	2208	2135	1983	1934	1948	2226	2202	1972	2097
TP	673	712	906	990	990	1029	1045	1052	1081	1019	1022	1021	1110	1938	1181	1181	1949	1475	1629	1164	1474	1540	1480	1933	1196	1835	1532	1487	1256	2086
1/0	665	695	884	983	982	1018	1017	1030	1067	1020	1020	1083	1172	1927	1166	1142	1939	1453	1604	1145	1444	1500	1464	1918	1165	1799	1493	1452	1254	2078
PMOVI	663	698	885	971	969	1015	1017	1032	1066	1015	1093	1156	1181	1936	1179	1166	1945	1468	1625	1152	1465	1511	1474	1925	1179	1806	1502	1462	1260	2085
CP	671	700	881	978	976	1015	1015	1029	1065	1074	1152	1137	1165	1917	1154	1136	1934	1448	1603	1129	1440	1492	1452	1912	1155	1793	1483	1444	1249	2070
AlgB	679	716	926	1027	1022	1071	1088	1116	1908	1917	1935	1924	1937	1983	1929	1926	1987	2135	2256	1923	2128	2210	2144	1988	1936	1952	2226	2203	1969	2103
B	682	719	915	1006	1002	1044	1069	1148	1940	1193	1209	1201	1206	1946	1189	1192	1957	1493	1652	1178	1493	1569	1499	1937	1214	1860	1566	1528	1263	2096
A	680	719	906	1000	1002	1032	1109	1188	1929	1168	1185	1175	1174	1935	1165	1165	1948	1470	1636	1150	1471	1536	1481	1927	1195	1846	1537	1494	1247	2086
CM	679	716	900	992	989	1092	1169	1196	1929	1151	1170	1157	1173	1931	1159	1168	1947	1462	1625	1148	1460	1517	1476	1931	1174	1832	1506	1465	1247	2082
MTSPF	656	683	890	983	1037	1140	1144	1183	1923	1135	1161	1138	1157	1926	1127	1117	1942	1450	1604	1114	1449	1477	1466	1924	1165	1786	1474	1425	1247	2074
MTSP	659	685	885	1047	1101	1147	1156	1189	1928	1143	1169	1147	1167	1924	1124	1131	1940	1462	1616	1134	1461	1478	1472	1926	1177	1791	1476	1434	1253	2077
MTS	651	678	937	1099	1084	1129	1140	1170	1919	1130	1145	1136	1141	1922	1125	1121	1942	1454	1612	1098	1450	1449	1462	1921	1164	1772	1447	1384	1241	2072
SCNP	661	727	986	1089	1081	1103	1117	1156	1919	1101	1122	1115	1125	1920	1104	1103	1930	1428	1589	1051	1419	1261	1445	1906	1135	1756	1252	1188	1232	2073
SCN	684	750	970	1072	1065	1097	1113	1150	1913	1087	1114	1102	1121	1912	1090	1082	1925	1420	1575	1041	1416	1217	1434	1902	1120	1711	1209	1144	1232	2067

TN6631 (1.80V / +145°C)

Table A.14. Union and intersection at TN6631

	SCN	SCNP	MTS	MTSP	MTSPP	CM	A	B	AlgB	CP	PMOVI	1/0	TP	U	X	Y	LR	LA	RAW	RAW1	AB	AB1	BDN	SR	SS	BLIF	Ham5R	Ham5W	G	HamWk
HamWk	82	89	97	102	95	105	105	102	116	111	111	108	107	116	97	106	117	111	115	106	110	52	109	116	106	29	56	16	109	128
G	78	90	95	97	96	105	106	105	110	111	107	108	109	113	102	106	111	111	110	106	109	48	107	111	104	21	53	14	120	139
Ham5W	9	8	11	13	13	14	13	13	16	14	14	14	14	16	13	13	16	16	39	14	16	30	15	16	14	16	40	41	147	153
Ham5R	46	47	50	52	51	53	51	50	54	54	54	53	53	55	50	52	56	56	80	53	55	55	55	55	53	16	82	83	149	154
BLIF	11	10	17	20	19	21	19	19	29	22	22	21	21	27	19	20	28	21	25	20	21	17	22	27	21	29	95	54	128	128
SS	83	90	97	97	96	108	102	104	106	113	114	108	109	107	97	107	112	113	113	107	112	48	112	106	115	123	144	142	131	137
SR	79	88	95	97	95	103	104	104	114	112	111	106	106	119	99	104	115	109	115	105	108	51	108	122	131	124	149	147	131	134
BDN	82	90	99	97	96	109	103	106	108	116	115	109	110	109	99	110	113	117	118	109	116	51	120	134	123	127	147	146	133	139
AB1	42	44	43	45	45	49	45	45	51	50	49	49	49	52	46	47	50	52	70	48	51	72	141	143	139	84	99	83	144	148
AB	82	92	99	98	97	109	103	106	109	116	115	112	112	109	99	111	112	118	120	111	121	142	125	135	124	129	148	146	132	139
RAW1	82	90	97	97	93	107	102	105	105	111	109	107	108	105	100	106	107	110	113	113	123	137	124	130	121	122	142	140	127	135
RAW	82	91	99	100	97	111	104	108	114	120	117	111	112	115	101	113	116	122	153	153	154	155	155	160	155	157	155	155	163	166
LA	83	92	99	97	96	111	105	108	110	117	115	111	114	112	99	112	114	126	157	129	129	146	129	139	128	134	152	151	135	143
LR	82	93	98	99	96	106	104	107	116	115	113	107	111	115	98	109	128	140	165	134	137	150	135	135	131	129	154	153	137	139
Y	80	89	96	101	98	105	103	106	108	113	110	106	106	106	101	116	135	130	156	123	126	141	126	134	124	125	146	144	130	138
X	74	83	91	95	93	100	98	99	101	103	99	99	99	100	104	119	134	131	156	117	126	130	125	127	122	114	136	132	122	135
U	80	90	96	99	97	106	106	105	116	113	112	109	109	123	127	133	136	137	161	131	135	143	134	126	131	125	150	148	130	135
TP	82	91	97	97	96	109	103	105	109	113	111	110	116	130	121	126	133	128	157	121	125	139	126	132	122	124	145	143	127	137
1/0	82	89	98	98	97	108	103	103	108	112	111	114	120	128	119	124	135	129	156	120	123	137	125	130	121	122	143	141	126	134
PMOVI	83	90	98	101	98	109	105	106	109	117	120	123	125	131	125	126	135	131	156	124	126	143	125	131	121	127	148	147	133	137
CP	82	92	98	101	99	111	105	109	114	123	126	125	126	133	124	126	136	132	156	125	128	145	127	133	125	130	151	150	132	140
AlgB	79	90	95	99	98	107	106	107	123	132	134	129	130	130	126	131	135	139	162	131	135	144	135	131	132	123	151	148	133	135
B	80	87	95	96	94	103	104	110	126	124	124	121	121	128	115	120	131	128	155	118	125	137	124	128	121	120	142	138	125	136
A	80	87	94	98	95	103	109	115	126	127	124	120	122	126	115	122	133	130	158	120	127	136	126	127	122	119	140	137	123	132
CM	82	88	96	98	96	114	120	121	130	126	125	120	121	131	118	125	136	129	156	120	126	137	125	133	121	122	143	141	129	137
MTSPF	74	82	89	95	99	117	113	115	124	123	121	116	119	125	110	117	131	129	155	119	123	126	123	126	118	109	130	127	123	132
MTSP	79	83	92	103	107	119	114	117	127	125	122	119	122	127	112	118	132	132	156	119	126	130	126	128	121	112	133	131	126	129
MTS	80	85	99	110	109	117	114	114	127	124	121	115	118	126	112	119	129	126	153	115	121	128	120	126	117	111	131	129	124	130
SCNP	77	94	108	114	111	120	116	117	127	125	124	119	119	127	115	121	129	128	156	117	123	122	124	128	119	113	129	127	124	133
SCN	83	100	102	107	108	115	112	113	127	124	120	115	117	126	113	119	129	126	154	114	122	113	121	126	115	101	119	115	125	129

TN3741 (1.30V / -40°C)

Table A.15. Union and intersection at TN3741

	SCN	SCNP	MTS	MTSP	MTSPP	CM	A	B	AlgB	CP	PMOVI	1/0	TP	U	X	Y	LR	LA	RAW	RAW1	AB	AB1	BDN	SR	SS	BLIF	Ham5R	Ham5W	G	HamWk
HamWk	76	90	79	92	93	94	95	99	102	109	109	102	105	104	91	103	94	131	147	103	141	124	138	98	111	60	121	99	117	154
G	78	97	82	102	106	105	109	119	120	128	124	120	124	117	101	120	104	124	145	124	133	118	137	107	137	66	114	75	159	196
Ham5W	63	64	55	64	67	68	66	70	70	73	73	70	70	69	69	71	65	102	110	75	109	109	109	68	75	52	112	113	197	168
Ham5R	70	85	70	87	92	91	88	98	94	101	104	97	102	95	90	100	86	135	147	107	139	146	142	92	110	63	163	164	208	196
BLIF	55	59	53	62	61	59	59	62	64	62	63	65	63	64	58	61	59	65	65	63	65	62	67	61	64	68	168	129	161	162
SS	78	94	79	102	101	104	107	119	114	123	120	115	121	115	98	116	99	120	139	116	128	114	130	109	145	149	198	183	167	188
SR	72	85	74	89	87	90	92	95	98	100	102	96	101	98	85	97	90	101	109	100	100	90	103	113	149	120	184	158	165	169
BDN	76	93	77	99	97	103	104	112	114	122	123	113	118	114	95	113	103	153	175	117	170	146	186	196	201	187	207	190	208	202
AB1	71	87	69	89	89	92	93	98	98	102	104	100	101	97	88	97	85	131	151	104	146	166	206	189	197	172	183	170	207	196
AB	75	92	78	99	97	101	103	113	113	122	116	114	117	114	95	112	101	149	172	113	184	204	200	197	201	187	208	188	210	197
RAW1	74	92	79	96	100	97	98	108	104	109	113	107	108	104	98	111	98	110	120	130	201	192	199	143	159	135	186	168	165	181
RAW	78	97	81	103	104	106	107	120	118	129	128	120	123	121	100	117	107	159	199	209	211	214	210	203	205	202	215	202	213	206
LA	75	90	79	94	98	96	101	111	109	110	114	105	112	109	95	106	97	165	205	185	200	200	198	177	190	168	193	176	200	188
LR	70	82	73	83	87	89	92	95	95	97	101	93	99	93	86	98	113	181	205	145	196	194	196	136	159	122	190	161	168	173
Y	75	89	78	94	100	95	98	107	105	108	112	107	112	104	97	124	139	183	206	143	196	193	197	140	153	131	187	166	163	175
X	72	80	75	90	93	86	86	92	93	93	95	92	94	97	103	130	130	173	202	135	192	181	194	131	150	113	176	147	161	166
U	77	90	78	97	96	90	94	105	104	111	105	103	108	125	131	145	145	181	203	151	195	194	197	140	155	129	193	169	167	175
TP	76	90	77	96	98	96	101	110	106	114	116	107	130	147	139	142	144	183	206	152	197	195	198	142	154	135	191	173	165	179
1/0	76	88	78	95	99	95	97	102	107	115	110	127	150	149	138	144	147	187	206	150	197	193	200	144	157	130	193	170	166	179
PMOVI	76	91	80	96	100	98	103	110	107	113	131	148	145	151	139	143	143	182	202	148	199	193	194	142	156	136	190	171	166	176
CP	76	90	75	96	96	98	101	107	111	134	152	146	150	148	144	150	150	189	204	155	196	198	198	147	156	140	196	174	165	179
AlgB	74	85	76	93	97	94	98	104	124	147	148	144	148	145	134	143	142	180	205	150	195	192	196	139	155	128	193	167	163	176
B	76	89	75	93	94	90	102	125	145	152	146	150	145	145	136	142	143	179	204	147	196	193	199	143	151	131	190	168	165	180
A	73	83	73	87	87	88	111	134	137	144	139	141	140	142	128	137	132	175	203	143	192	184	193	132	149	120	186	158	161	170
CM	69	84	74	88	90	109	132	144	139	145	142	141	143	144	126	138	133	178	202	142	192	183	192	132	150	118	181	154	163	169
MTSPF	73	81	77	88	107	126	131	138	134	145	138	135	139	136	117	131	133	174	202	137	194	184	196	133	151	114	178	153	160	168
MTSP	70	83	72	105	124	126	129	137	136	143	140	137	139	133	118	135	135	176	201	139	190	182	192	129	148	111	181	154	162	167
MTS	65	74	82	115	112	117	120	132	130	141	133	131	135	129	110	128	122	168	200	133	188	179	191	121	148	97	175	140	159	157
SCNP	74	98	106	120	124	123	126	134	137	142	138	137	138	133	121	133	129	173	200	136	190	177	191	126	149	107	176	147	160	162
SCN	79	103	96	114	113	119	117	128	129	137	134	130	133	127	110	128	122	169	200	135	188	174	189	120	146	92	172	129	160	157

TN3841 (1.50V / -40°C)

Table A.16. Union and intersection at TN3841

SCN	SCNP	MTS	MTSP	MTSPF	CM	A	B	AlgB	CP	PMOVI	1/0	TP	U	X	Y	LR	LA	RAW	RAW1	AB	AB1	BDN	SR	SS	BLIF	Ham5R	Ham5W	G	HamWk	
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	9	9	9	10	10	8	9	9	10	10	SCN
13	**13**	10	10	11	11	10	11	11	10	11	12	10	11	11	11	11	12	11	11	10	9	10	12	12	9	10	9	11	11	SCNP
10	13	**10**	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	9	9	9	10	10	8	9	9	10	10	MTS
13	16	13	**13**	13	12	11	12	11	12	12	13	12	12	11	13	11	12	12	12	12	11	11	12	13	8	12	11	12	11	MTSP
16	18	16	16	**16**	13	11	14	13	13	14	16	13	14	12	16	13	14	15	14	14	12	13	14	16	9	13	11	15	12	MTSPP
13	15	13	14	16	**13**	11	13	12	12	13	13	12	13	12	13	12	13	13	13	12	10	12	12	13	9	12	11	13	12	CM
11	14	11	13	16	13	**11**	11	11	11	11	11	11	11	10	11	11	11	11	11	10	9	10	11	11	8	10	10	11	11	A
15	17	15	16	17	15	15	**15**	12	14	15	14	13	14	12	14	13	14	15	13	14	10	14	13	15	9	12	11	15	12	B
14	16	14	16	17	15	14	17	**14**	11	12	13	11	12	11	14	12	14	14	13	12	11	12	12	13	9	12	11	14	13	AlgB
14	17	14	15	17	15	14	15	17	**14**	14	13	13	13	11	13	12	13	14	12	13	10	13	12	14	8	11	11	14	11	CP
16	18	16	17	18	16	16	16	18	16	**16**	15	13	14	12	14	13	15	16	13	15	10	15	13	15	9	12	11	15	12	PMOVI
18	19	18	18	18	18	18	19	19	19	19	**18**	13	14	12	16	13	15	16	14	15	12	14	15	16	9	13	11	15	12	1/0
13	16	13	14	16	14	13	15	16	14	16	18	**13**	13	11	13	12	12	13	12	12	10	12	12	13	8	11	11	13	11	TP
14	16	14	15	16	14	14	15	16	15	16	18	14	**14**	12	14	13	13	14	13	13	10	13	13	14	9	12	11	14	12	U
12	14	12	14	16	13	13	15	15	15	16	18	14	14	**12**	12	11	12	12	12	11	10	11	11	12	9	11	10	12	11	X
17	19	17	17	17	17	17	18	17	18	19	19	17	17	17	**17**	13	15	16	14	15	13	14	14	16	9	14	12	16	13	Y
13	15	13	15	16	14	13	15	15	15	16	18	14	14	14	17	**13**	12	13	12	12	9	12	13	13	9	11	10	13	12	LR
56	57	56	57	58	56	56	57	56	57	57	59	57	57	56	58	57	**56**	55	14	52	50	52	12	17	9	51	48	16	40	LA
62	64	62	63	63	62	62	62	62	62	62	64	62	62	62	63	62	63	**62**	14	55	55	53	13	17	9	55	49	17	41	RAW
14	16	14	15	16	14	14	16	15	16	17	18	15	15	14	17	15	56	62	**14**	12	11	12	12	14	9	12	11	14	12	RAW1
57	59	57	57	58	57	57	57	58	57	57	59	57	57	57	58	57	60	63	58	**56**	50	53	13	16	9	52	48	15	40	AB
61	64	61	62	64	63	62	65	63	64	66	66	63	64	62	64	64	66	67	63	66	**60**	48	10	13	8	56	49	12	38	AB1
54	56	54	55	56	54	54	54	55	54	54	57	54	54	54	56	54	57	62	55	56	65	**53**	12	15	9	50	48	15	39	BDN
15	16	15	16	17	16	15	17	17	17	18	18	16	16	16	18	15	59	64	17	58	65	56	**15**	14	9	12	10	13	12	SR
19	20	19	19	19	19	19	19	20	19	20	21	19	19	19	20	19	58	64	19	59	66	57	20	**19**	9	14	12	16	13	SS
11	13	11	14	16	13	12	15	14	15	16	18	14	14	12	17	13	56	62	14	56	61	53	15	19	**9**	9	8	9	9	BLIF
59	61	59	59	61	59	59	61	60	61	62	63	60	60	59	61	60	63	65	60	62	62	61	61	63	58	**58**	50	13	40	Ham5R
51	54	51	52	55	52	51	54	53	53	55	57	52	53	52	55	53	58	63	53	58	61	55	55	57	51	58	**50**	12	38	Ham5W
17	19	17	18	18	17	17	17	17	17	18	20	17	17	17	18	17	57	62	17	58	65	55	19	20	17	62	55	**17**	13	G
41	43	41	43	45	42	41	44	42	44	45	47	43	43	42	45	42	57	62	43	57	63	55	44	47	41	59	53	45	**41**	HamWk

TN3941 (1.80V / -40°C)

Table A.17. Union and intersection at TN3941

	SCN	SCNP	MTS	MTSP	MTSPP	CM	A	B	AlgB	CP	PMOVI	1/0	TP	U	X	Y	LR	LA	RAW	RAW1	AB	AB1	BDN	SR	SS	BLIF	Ham5R	Ham5W	G	HamWk
HamWk	21	20	20	19	19	20	20	22	20	21	21	20	20	22	19	19	21	26	25	20	27	24	26	22	20	19	24	20	22	28
G	21	20	20	19	19	20	20	21	20	22	22	20	20	21	19	19	20	22	21	20	22	20	21	22	21	19	19	16	23	29
Ham5W	16	16	16	17	16	16	16	16	16	16	16	17	16	16	17	16	16	28	30	17	27	28	27	16	17	16	31	32	39	40
Ham5R	19	19	19	19	18	19	19	19	19	19	19	20	19	19	19	18	19	33	35	19	34	35	33	19	20	19	40	41	44	44
BLIF	19	19	19	18	18	19	19	19	19	19	19	19	19	19	18	18	19	19	19	18	19	19	19	19	19	19	40	35	23	28
SS	20	20	20	20	19	20	20	20	20	21	21	21	20	20	20	19	20	22	22	20	21	21	21	20	22	22	42	37	24	30
SR	21	20	20	19	19	20	20	21	20	21	21	20	20	21	19	19	20	21	20	20	22	20	21	22	24	22	43	38	23	28
BDN	20	20	20	20	19	20	20	21	20	20	20	21	20	21	20	19	20	33	33	20	34	29	35	36	36	35	42	40	37	37
AB1	20	20	20	20	19	20	20	20	20	20	20	21	20	20	20	19	20	29	31	20	30	37	43	39	38	37	42	41	40	41
AB	21	20	20	20	19	20	20	21	20	21	21	21	20	21	20	19	20	34	33	21	37	44	38	37	38	37	43	42	38	38
RAW1	20	19	19	20	19	19	19	19	19	20	20	20	19	19	20	19	19	21	20	21	37	38	36	23	23	22	42	36	24	29
RAW	20	20	20	20	19	20	20	20	20	21	21	21	20	20	20	19	20	35	37	38	41	43	39	39	37	37	42	39	39	40
LA	21	20	20	20	19	20	20	20	20	22	22	21	20	20	20	19	20	36	38	36	39	44	38	37	36	36	43	40	37	38
LR	20	20	20	19	19	20	20	21	20	20	20	20	20	21	19	19	21	37	38	23	38	38	36	23	23	21	42	37	24	28
Y	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19	21	36	37	21	37	37	35	22	22	20	41	35	23	28
X	19	19	19	20	19	19	19	19	19	19	19	20	19	19	20	20	22	36	37	21	37	37	35	23	22	21	41	35	24	29
U	20	20	20	19	19	20	20	22	20	20	20	20	20	22	23	22	22	38	39	24	38	39	36	23	24	22	43	38	24	28
TP	20	20	20	19	19	20	20	20	20	20	20	20	20	22	21	20	21	36	37	22	37	37	35	22	22	20	41	36	23	28
1/0	20	20	20	20	19	20	20	20	20	20	20	21	21	23	21	21	22	36	37	22	37	37	35	23	22	21	41	36	24	29
PMOVI	21	20	20	19	19	20	20	20	20	22	22	23	22	24	23	22	23	36	38	23	38	39	37	23	23	22	43	38	23	29
CP	21	20	20	19	19	20	20	20	20	22	22	23	22	24	23	22	23	36	38	23	38	39	37	23	23	22	43	38	23	29
AlgB	20	20	20	19	19	20	20	20	20	22	22	21	20	22	21	20	21	36	37	22	37	37	35	22	22	20	41	36	23	28
B	20	20	20	19	19	20	20	22	22	24	24	23	22	22	23	22	22	38	39	24	38	39	36	23	24	22	43	38	24	28
A	20	20	20	19	19	20	20	22	20	22	22	21	20	22	21	20	21	36	37	22	37	37	35	22	22	20	41	36	23	28
CM	20	20	20	19	19	20	20	22	20	22	22	21	20	22	21	20	21	36	37	22	37	37	35	22	22	20	41	36	23	28
MTSPF	19	19	19	19	19	20	20	22	20	22	22	21	20	22	20	19	21	36	37	21	37	37	35	22	22	20	41	35	23	28
MTSP	19	19	19	20	20	21	21	23	21	23	23	21	21	23	20	20	22	36	37	21	37	37	35	23	22	21	41	35	24	29
MTS	20	20	20	21	20	20	20	22	20	22	22	21	20	22	21	20	21	36	37	22	37	37	35	22	22	20	41	36	23	28
SCNP	20	20	20	21	20	20	20	22	20	22	22	21	20	22	21	20	21	36	37	22	37	37	35	22	22	20	41	36	23	28
SCN	21	21	21	22	21	21	21	23	21	22	22	22	21	23	22	21	22	36	38	22	37	38	36	22	23	21	42	37	23	28

TN4441 (1.30V / +25°C)

Table A.18. Union and intersection at TN4441

	SCN	SCNP	MTS	MTSP	MTSPP	CM	A	B	AlgB	CP	PMOVI	1/0	TP	U	X	Y	LR	LA	RAW	RAW1	AB	AB1	BDN	SR	SS	BLIF	Ham5R	Ham5W	G	HamWk
HamWk	26	34	31	34	38	39	37	40	40	42	39	40	36	42	33	39	43	86	86	41	84	80	85	41	42	21	80	71	45	**95**
G	30	36	31	38	44	45	45	48	48	49	50	47	42	48	42	45	47	53	55	50	52	48	51	46	49	23	50	34	**63**	113
Ham5W	22	25	24	30	31	31	31	32	33	33	37	35	31	34	29	35	30	86	96	34	90	92	87	32	38	21	96	**98**	127	122
Ham5R	30	32	28	37	41	40	39	45	48	46	49	52	40	46	40	45	41	104	119	46	109	113	108	41	49	24	**130**	132	143	145
BLIF	13	20	17	19	20	21	21	21	22	21	23	22	20	20	20	20	20	22	24	22	22	23	23	21	23	**24**	130	101	64	98
SS	31	34	31	39	44	43	45	43	48	49	51	48	42	48	40	44	44	54	58	44	56	51	53	43	**63**	64	144	123	77	116
SR	32	35	31	35	41	38	40	45	42	42	45	40	41	48	33	40	46	46	48	43	46	40	46	**53**	73	56	142	119	70	107
BDN	30	35	31	40	45	43	45	44	50	49	53	49	42	50	40	47	45	104	112	45	107	102	**120**	127	130	121	142	131	132	130
AB1	30	31	27	34	40	41	39	42	43	47	46	47	39	45	38	44	39	99	111	45	106	**118**	136	131	130	119	135	124	133	133
AB	32	35	31	38	48	45	46	48	51	52	51	52	45	50	41	49	46	107	118	49	**123**	135	136	130	130	125	144	131	134	134
RAW1	30	35	30	37	40	41	39	47	43	44	45	45	40	44	39	42	44	48	49	**54**	128	127	129	64	73	56	138	118	67	108
RAW	33	35	31	40	48	44	47	49	51	54	55	56	46	55	42	48	49	112	**139**	144	144	146	147	144	144	139	150	141	147	148
LA	30	35	31	39	44	43	45	46	49	50	53	52	43	49	40	49	50	**118**	145	124	134	137	134	125	127	120	144	130	128	127
LR	30	34	30	35	41	40	40	44	42	44	45	43	43	45	35	42	**55**	123	145	65	132	134	130	62	74	59	144	123	71	107
Y	28	32	29	37	40	39	39	43	43	46	45	42	41	43	36	**51**	64	120	142	63	125	125	124	64	70	55	136	114	69	107
X	26	28	25	33	35	37	36	36	39	37	39	39	35	36	**44**	59	64	122	141	59	126	124	124	64	67	48	134	113	65	106
U	32	34	31	37	40	41	43	46	44	44	46	42	41	**61**	69	69	71	130	145	71	134	134	131	66	76	65	145	125	76	114
TP	27	31	28	33	38	39	36	41	40	41	43	41	**49**	69	58	59	61	124	142	63	127	128	127	61	70	53	139	116	70	108
1/0	29	32	28	38	42	38	38	43	46	45	47	**58**	66	77	63	67	70	124	141	67	129	129	129	71	73	60	136	121	74	113
PMOVI	30	34	31	36	44	38	43	45	47	47	**57**	68	63	72	62	63	67	122	141	66	129	129	124	65	69	58	138	118	70	113
CP	31	34	31	37	44	41	44	45	47	**56**	66	69	64	73	63	61	67	124	141	66	127	127	127	67	70	59	140	121	70	109
AlgB	28	33	30	37	43	39	43	43	**56**	65	66	68	65	73	61	64	69	125	144	67	128	131	126	67	71	58	138	121	71	111
B	32	36	31	35	40	39	38	**51**	64	62	63	66	59	66	59	59	62	123	141	58	126	127	127	59	71	54	136	117	66	106
A	27	32	31	33	40	39	**49**	62	62	61	63	69	62	67	57	61	64	122	141	64	126	128	124	62	67	52	140	116	67	107
CM	27	32	28	35	39	**51**	61	63	68	66	70	71	61	71	58	63	66	126	146	64	129	128	128	66	71	54	141	118	69	107
MTSPF	28	33	29	35	**50**	62	59	61	63	62	63	66	61	71	59	61	64	124	141	64	125	128	125	62	69	54	139	117	69	107
MTSP	25	29	26	**44**	59	60	60	60	63	63	65	64	60	68	55	58	64	123	143	61	129	128	124	62	68	49	137	112	69	105
MTS	25	30	**31**	49	52	54	49	51	57	56	57	61	52	61	50	53	56	118	139	55	123	122	120	53	63	38	133	105	63	95
SCNP	25	**36**	37	51	53	55	53	51	59	58	59	62	54	63	52	55	57	119	140	55	124	123	121	54	65	40	134	109	63	97
SCN	**34**	45	40	53	56	58	56	53	62	59	61	63	56	63	52	57	59	122	140	58	125	122	124	55	66	45	134	110	67	103

TN4541 (1.80V / +25°C)

Table A.19. Union and intersection at TN4541

	SCN	SCNP	MTS	MTSP	MTSPP	CM	A	B	AlgB	CP	PMOVI	1/0	TP	U	X	Y	LR	LA	RAW	RAW1	AB	AB1	BDN	SR	SS	BLIF	Ham5R	Ham5W	G	HamWk
HamWk	6	7	4	4	5	6	7	7	7	7	6	7	7	8	5	5	8	10	9	4	9	6	9	8	7	3	7	6	7	**10**
G	6	6	4	4	5	5	6	6	6	7	5	6	6	8	4	4	7	8	7	5	7	5	7	7	7	3	5	4	**8**	11
Ham5W	4	4	3	3	3	4	4	4	4	4	4	4	4	4	3	3	4	13	16	3	13	13	10	4	4	3	17	**18**	22	22
Ham5R	4	5	3	3	3	4	5	5	5	5	4	5	5	5	3	3	5	14	17	3	14	12	11	5	5	3	**18**	19	21	21
BLIF	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	**3**	18	18	8	10
SS	7	7	4	4	6	6	7	7	7	8	6	7	7	8	5	5	7	8	8	5	8	5	8	7	**8**	8	21	22	9	11
SR	6	7	4	4	5	6	7	7	7	7	6	7	7	8	5	5	8	8	7	4	7	4	7	**8**	9	8	21	22	9	10
BDN	7	7	4	4	6	6	7	7	7	8	6	7	7	8	5	5	7	14	14	5	14	9	**14**	15	14	14	21	22	15	15
AB1	5	4	4	4	5	4	4	4	4	5	4	4	4	5	4	4	4	12	15	5	11	**17**	22	21	20	17	23	22	20	21
AB	7	7	4	4	6	6	7	7	7	8	6	7	7	8	5	5	7	17	17	5	**17**	23	17	18	17	17	21	22	18	18
RAW1	5	4	4	4	5	4	4	4	4	5	4	4	4	5	4	4	4	5	5	**5**	17	17	14	9	8	5	20	20	8	11
RAW	7	7	4	4	6	6	7	7	7	8	6	7	7	8	5	5	7	17	**21**	21	21	23	21	22	21	21	22	23	22	22
LA	7	7	4	4	6	6	7	7	7	8	6	7	7	9	5	5	8	**19**	23	19	19	24	19	19	19	19	23	24	19	19
LR	6	7	4	4	5	6	7	7	7	7	6	7	7	8	5	5	**8**	19	22	9	18	21	15	8	9	8	21	22	9	10
Y	5	5	4	4	5	5	5	5	5	5	5	5	5	5	5	**5**	8	19	21	6	17	18	14	8	8	5	20	20	9	10
X	5	5	4	4	5	5	5	5	5	5	5	5	5	5	**5**	5	8	19	21	6	17	18	14	8	8	5	20	20	9	10
U	7	7	4	4	6	6	7	7	7	8	6	7	7	**9**	9	9	9	19	22	9	18	21	15	9	9	9	22	23	9	11
TP	6	7	4	4	5	6	7	7	7	7	6	7	**7**	9	7	7	8	19	21	8	17	20	14	8	8	7	20	21	9	10
1/0	6	7	4	4	5	6	7	7	7	7	6	**7**	7	9	7	7	8	19	21	8	17	20	14	8	8	7	20	21	9	10
PMOVI	6	6	4	4	5	6	6	6	6	6	**6**	7	7	9	6	6	8	19	21	7	17	19	14	8	8	6	20	20	9	10
CP	7	7	4	4	6	6	7	7	7	**8**	8	8	8	9	8	8	9	19	21	8	17	20	14	9	8	8	21	22	9	11
AlgB	6	7	4	4	5	6	7	7	**7**	8	7	7	7	9	7	7	8	19	21	8	17	20	14	8	8	7	20	21	9	10
B	6	7	4	4	5	6	7	**7**	7	8	7	7	7	9	7	7	8	19	21	8	17	20	14	8	8	7	20	21	9	10
A	6	7	4	4	5	6	**7**	7	7	8	7	7	7	9	7	7	8	19	21	8	17	20	14	8	8	7	20	21	9	10
CM	6	6	4	4	5	**6**	7	7	7	8	6	7	7	9	6	6	8	19	21	7	17	19	14	8	8	6	20	20	9	10
MTSPF	6	5	4	4	**6**	7	8	8	8	8	7	8	8	9	6	6	9	19	21	6	17	18	14	9	8	6	21	21	9	11
MTSP	4	4	4	**4**	6	6	7	7	7	8	6	7	7	9	5	5	8	19	21	5	17	17	14	8	8	4	19	19	8	10
MTS	4	4	**4**	4	6	6	7	7	7	8	6	7	7	9	5	5	8	19	21	5	17	17	14	8	8	4	19	19	8	10
SCNP	6	**7**	7	7	8	7	7	7	7	8	7	7	7	9	7	7	8	19	21	8	17	20	14	8	8	7	20	21	9	10
SCN	**7**	8	7	7	7	7	8	8	8	8	7	8	8	9	7	7	9	19	21	7	17	19	14	9	8	7	21	21	9	11

Appendix B
Calculation of Fault Distribution

From the data based, the following values for the cardinality of sets have been determined:

$|F_0| = 2439$
$|F_1| = 1610$
$|F_2| = 1073$
$|F_3| = 390$
$|F_4| = 171$
$|F_5| = 0$

So is for the number of different faults per fault modes:

$$|\{SS\}| = |F_0| - |F_1| = 2439 - 1610 = 829 \quad (39)$$

$$|\{\text{some CF}\}| = |F_1| - |F_2| = 1610 - 1073 = 537 \quad (40)$$

$$|\{\text{remaining CF}\}| = |F_3| - |F_1| = 1073 - 390 = 683 \quad (41)$$

$$|\{LF\}| = |F_3| - |F_4| = 390 - 171 = 219 \quad (42)$$

$$|\{DF\}| = |F_4| - |F_5| = 171 - 0 = 171 \quad (43)$$

Appendix C
ESPRESSO Algorithm and Software

C.1 ESPRESSO Heuristic Algorithm

ESPRESSO is a heuristic logic minimization algorithm developed for the optimization of digital logic gate circuits. ESPRESSO was developed early 1980s at IBM by Robert K. Brayton [50].

Classical minimization methods are the use of Karnaugh-Maps or the Quine-McCluskey Algorithm. However, the use of a Karnough map is reasonable for small systems with few inputs, and if the minimization could be done manually. For large systems with many inputs, Quine-McCluskey would be feasible but ESPRESSO is much more efficient due to computation time.

The ESPRESSO is a relatively complex algorithm that maps the truth table of a system into a geometrical representation of n-dimensional hyper cubes, where n is the number of inputs to the system. Three sets of input values are distinguished:

- The On-Set, where the output is TRUE
- the Off-Set, where the output is FALSE, and
- the DC-Set, where the output is don't care.

Seven basic routines are involved into ESPRESSO-II minimization [50, 51]: COMPLEMEN, EXPAND, ESSENTIAL_PRIMES, IRREDUNDANT_COVER, REDUCE, LAST_GASP, MAKE_SPARES.

The ESPRESSO algorithm will not be explained in this work. For the computation of the optimal test sets, ESPRESSO was used in the software package “Logic Friday” from [52].

C.2 ESPRESSO Software

To compute the optimal test sets, ESPRESSO was used from the “Logic Friday” software package [52]. This section briefly describes the input and output format [53, 54] for the data.

```
.i 30
.o 1
.type r
--------0----0--0-0--0-00000-0 0
000000000000000000000-000---00 0
--0000000000000000000-0000--00 0
-----------------00-000---00-- 0
0000000000000000000000000-0-00 0
--0000000000000000000-0000--00 0
--------0----0--0------0-0---0 0
------------------0--0--0-00-- 0
---000000000000000000-0000--00 0
--------0----0--0------0-0--00 0
--------0-------0------------0 0
---000000000000000000000000000 0
---00---0--0-0-00-0----0-0--00 0
--00000000-0000000000-0-0---00 0
--0000000000000000000-0000--00 0
--------0-------0------0-0---0 0
--------0----0--0------000---0 0
-------------0---------------- 0
----------------0------------0 0
--------0----0--0------0-0--00 0
000000000000000000000-0        0
00---000000000--0000
    ----0----0--
```

```
.i 30
.o 1
.p 4
-------1-----1--1111-1---11-11 1
-------1-----11-111--1---11-11 1
-------1-----1--1111-1-1--1-11 1
-------1-----11-111--1-1--1-11 1
.e
```

Figure C.1. Input and output file to ESPRESSO

Figure C.1. shows the input (left) and output file of ESPRESSO, where

- `.i 30` specifies the number of input variables.
- `.o 1` specifies the number of output variables.
- `.type r` specifies the OFF-Set as input.
- `.p 4` specifies the number of results
- `.e` denotes the end of the file.

Both, input and output are text files. The input file contains the truth table to be minimized, the output file the minimized function. The 30 inputs are separated by a 'space' from the output. The sequence of inputs represents the sequence of algorithms A_1 trough A_{30} as given in table 4.2. The truth table is in disjunctive normal form (DNF).

The output file also contains the result of the minimization as truth table with inputs and output space-separated. A '1' denotes that the corresponding algorithm A_m is essential. If the table consists of more than one row, the solutions are equivalent.

Appendix D
Additional Results of Chapter 7

For all test numbers the essential algorithms are determined and the relation of FC and TL is done. It is remarkable that for any test condition, at least one of the algorithms March RAW, Ham5R or Ham_Walk is essential and occurs on top of the list of weighted algorithms.

Hence, for productive testing, a combination of these algorithms works well with any environmental test condition.

TN1522 (1.35V / +125°C)

Table D.1. Fault coverage over test length at TN1522

#	Algorithm	∆FC	∆TL	FC	TL
1	March RAW	447	26n	72.45%	26n
2	Ham_Walk	133	15n	94.00%	41n
3	Ham5R	20	25n	97.24%	66n
4	March LA	5	22n	98.06%	88n
5	March U	3	14n	98.54%	102n
6	March AB1	3	11n	99.03%	113n
7	March SS	3	22n	99.51%	135n
8	Ham5W	2	25n	99.84%	160n
9	March A	1	15n	100%	175n

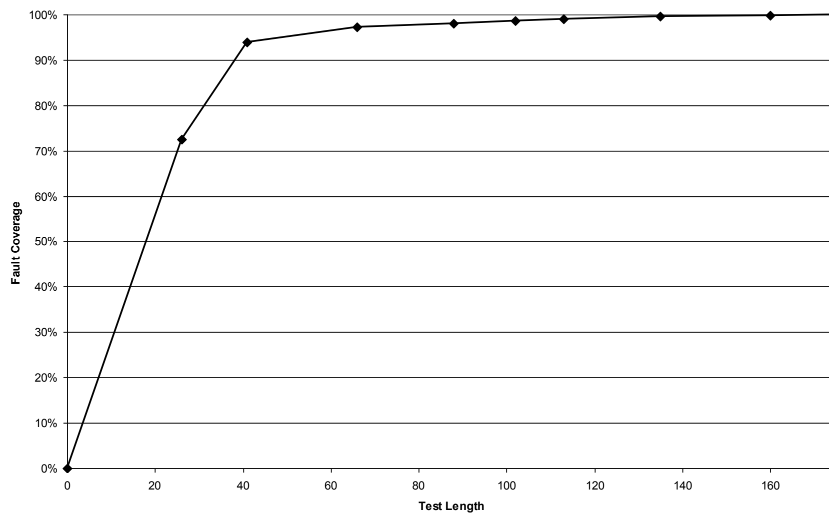

Figure D.1. Fault coverage over test length at TN1522

$$TT_{1522} = \frac{175 \cdot 261{,}56\text{kB} \cdot \dfrac{1024 \cdot 8}{32\text{kB}}}{180\text{MHz}} = 65.1\text{ms} \tag{44}$$

TN1622 (1.80V / +125°C)

Table D.2. Fault coverage over test length at TN1622

#	Algorithm	ΔFC	ΔTL	FC	TL
1	March RAW	46	26n	82.14%	46n
2	Ham_Walk	7	15n	94.64%	41n
3	March AB1	3	11n	100%	52n

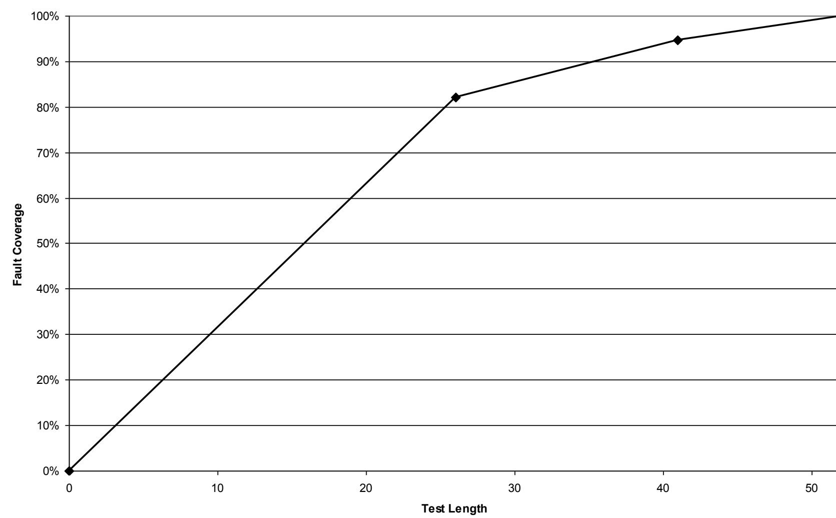

Figure D.2. Fault coverage over test length at TN1622

$$TT_{1622} = \frac{52 \cdot 261{,}56\text{kB} \cdot \frac{1024 \cdot 8}{32\text{kB}}}{180\text{MHz}} = 19.3\text{ms} \tag{45}$$

TN6531 (1.35V / +140°C)

Table D.3. Fault coverage over test length at TN6531

#	Algorithm	ΔFC	ΔTL	FC	TL
1	Ham_Walk	2063	15n	84.58%	15n
2	March RAW	284	26n	96.23%	41n
3	Ham5R	36	25n	97.70%	66n
4	March G	30	23n+2*D*	98.93%	89n+2*D*
5	March AB1	13	11n	99.47%	100+2*D*
6	March LR	5	14n	99.67%	114+2*D*
7	March X	3	6n	99.79%	120+2*D*
8	March LA	2	22n	99.88%	142+2*D*
9	March U	1	14n	99.92%	156+2*D*
10	March B	1	14n	99.96%	170+2*D*
11	March SR	1	17n	100%	187+2*D*

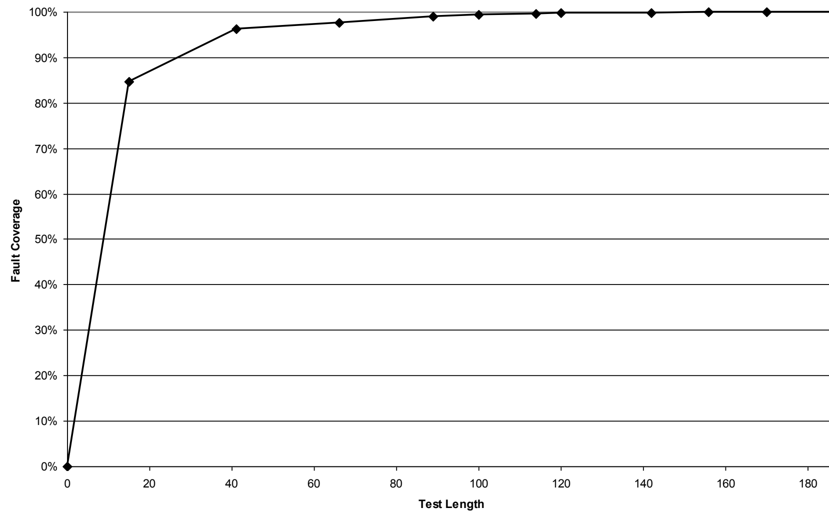

Figure D.3. Fault coverage over test length at TN6531

$$TT_{6531} = \frac{187 \cdot 261{,}56\text{kB} \cdot \dfrac{1024 \cdot 8}{32\text{kB}}}{180\text{MHz}} + 2 \cdot 100\text{ms} = 269.6\text{ms} \tag{46}$$

TN6631 (1.80V / +140°C)

Table D.4. Fault coverage over test length at TN6631

#	Algorithm	ΔFC	ΔTL	FC	TL
1	March RAW	153	26n	87.43%	26n
2	Ham_Walk	13	15n	94.86%	41n
3	March G	5	23n+2*D*	97.71%	64n+2*D*
4	March AB1	2	11n	98.86%	75n+2*D*
5	March LR	2	14n	100%	89n+2*D*

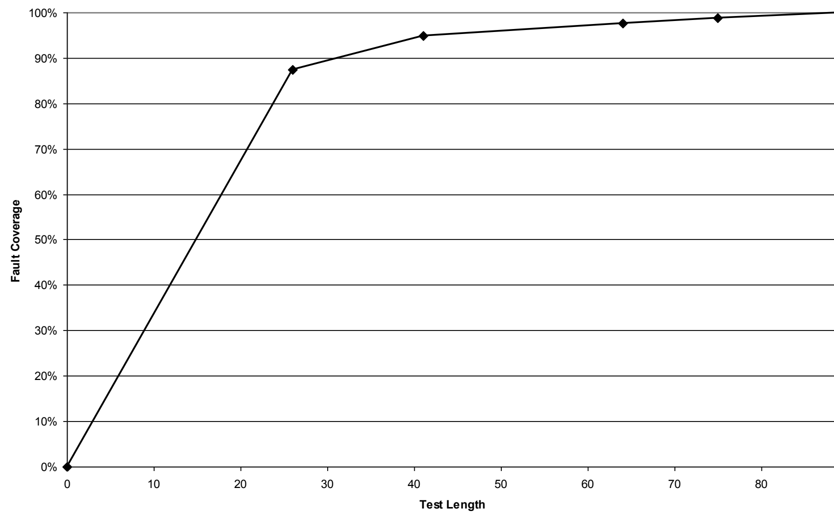

Figure D.4. Fault coverage over test length at TN6631

$$TT_{6631} = \frac{89 \cdot 261{,}56\text{kB} \cdot \frac{1024 \cdot 8}{32\text{kB}}}{180\text{MHz}} + 2 \cdot 100\text{ms} = 233.1\text{ms} \quad (47)$$

TN3741 (1.30V / -40°C)

Table D.5. Fault coverage over test length at TN3741

#	Algorithm	ΔFC	ΔTL	FC	TL
1	March RAW	199	26n	83.97%	26n
2	Ham5R	16	25n	90.72%	51n
3	March AB	7	22n	93.67%	73n
4	March AB1	4	11n	95.36%	84n
5	March SR	3	14n	96.62%	98n
6	March RAW1	2	13n	97.47%	111n
7	March BDN	2	22n	98.31%	133n
8	March TP	1	11n	98.73%	144n
9	Ham_Walk	1	15n	99.16%	159n
10	Algorithm B	1	17n	99.58%	176n
11	March G	1	23n+2*D*	100%	199n+2*D*

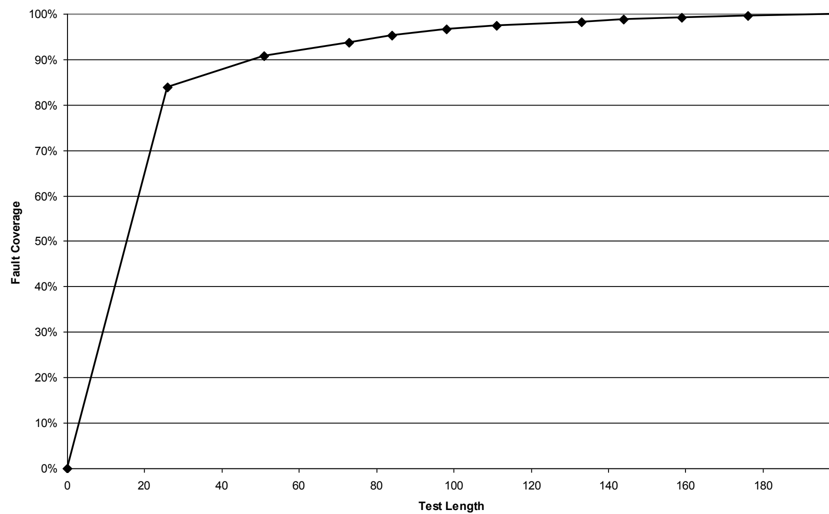

Figure D.5. Fault coverage over test length at TN3741

$$TT_{3741} = \frac{199 \cdot 261{,}56\text{kB} \cdot \dfrac{1024 \cdot 8}{32\text{kB}}}{180\text{MHz}} + 2 \cdot 100\text{ms} = 274.0\text{ms} \tag{48}$$

TN3841 (1.50V / -40°C)

Table D.6. Fault coverage over test length at TN3841

#	Algorithm	ΔFC	ΔTL	FC	TL
1	March RAW	62	26n	89.86%	26n
2	March AB1	5	11n	97.10%	37n
3	SCAN+	2	8n	100%	45n

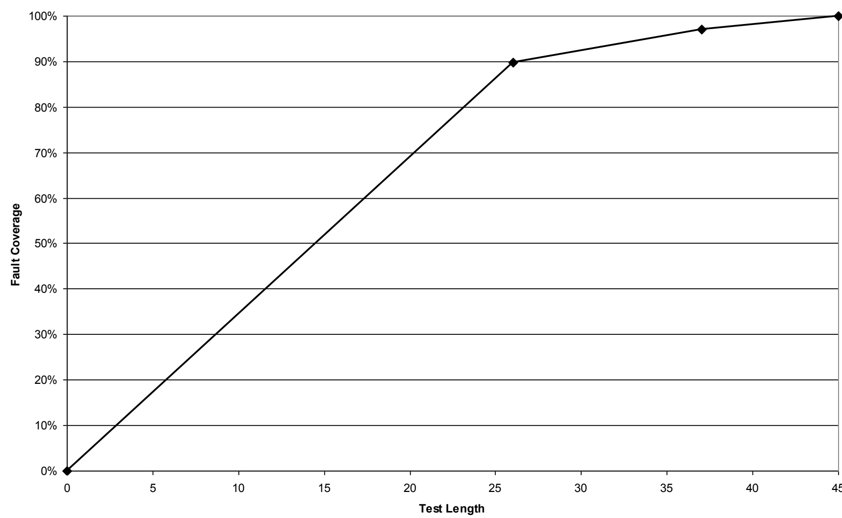

Figure D.6. Fault coverage over test length at TN3841

$$TT_{3841} = \frac{45 \cdot 261{,}56\text{kB} \cdot \frac{1024 \cdot 8}{32\text{kB}}}{180\text{MHz}} = 16.7\text{ms} \tag{49}$$

TN3941 (1.80V / -40°C)

Table D.7. Fault coverage over test length at TN3941

#	Algorithm	ΔFC	ΔTL	FC	TL
1	Ham5R	40	25n	86.96%	25n
2	Ham_Walk	3	15n	93.48%	40n
3	March RAW	2	26n	97.83%	66n
4	Ham5W	1	25n	100%	91n

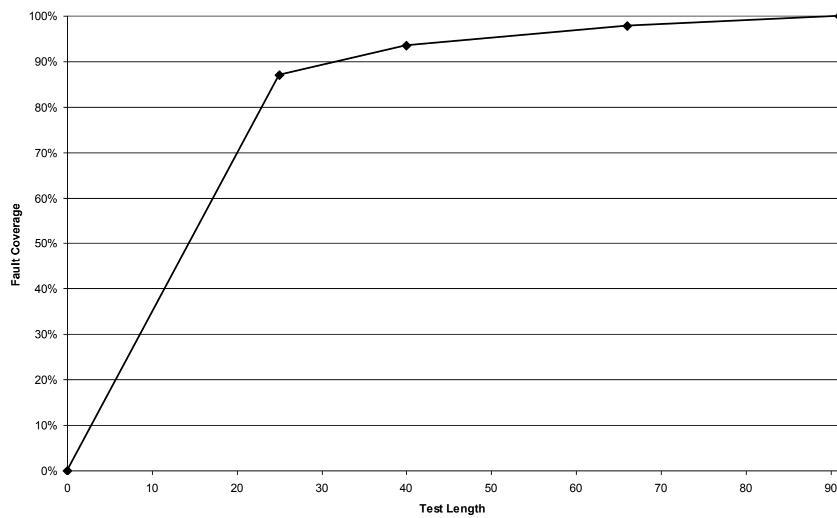

Figure D.7. Fault coverage over test length at TN3941

$$TT_{3941} = \frac{91 \cdot 261{,}56\text{kB} \cdot \frac{1024 \cdot 8}{32\text{kB}}}{180\text{MHz}} = 33.9\text{ms} \qquad (50)$$

TN4441 (1.30V / +25°C)

Table D.8. Fault coverage over test length at TN4441

#	Algorithm	ΔFC	ΔTL	FC	TL
1	March RAW	139	26n	84.76%	26n
2	Ham5R	11	25n	91.46%	51n
3	Ham_Walk	7	15n	95.73%	66n
4	March C-	4	10n	98.17%	76n
5	MATS+	2	5n	99.39%	81n
6	MATS++	1	6n	100%	87n

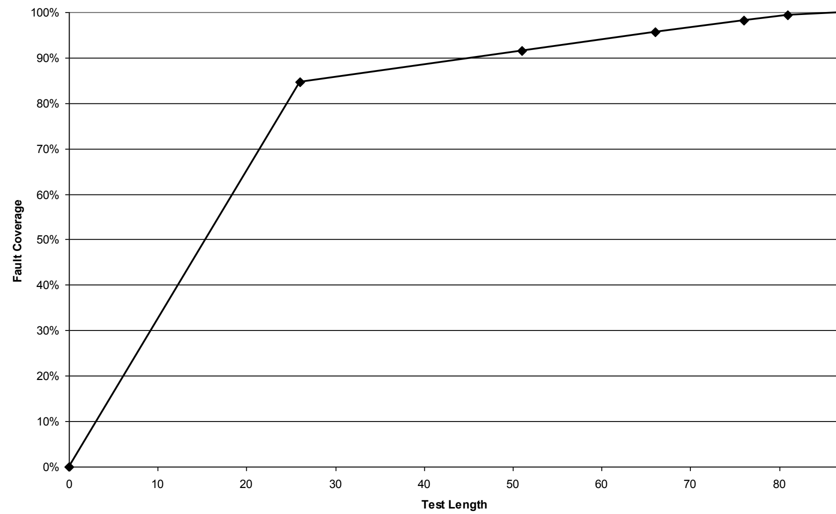

Figure D.8. Fault coverage over test length at TN4441

$$TT_{4441} = \frac{87 \cdot 261{,}56\text{kB} \cdot \dfrac{1024 \cdot 8}{32\text{kB}}}{180\text{MHz}} = 32.4\text{ms} \qquad (51)$$

TN4541 (1.80V / +25°C)

Table D.9. Fault coverage over test length at TN4541

#	Algorithm	∆FC	∆TL	FC	TL
1	Ham5R	18	25n	72.00%	25n
2	March AB1	5	11n	92.00%	36n
3	March SR	2	14n	100%	50n

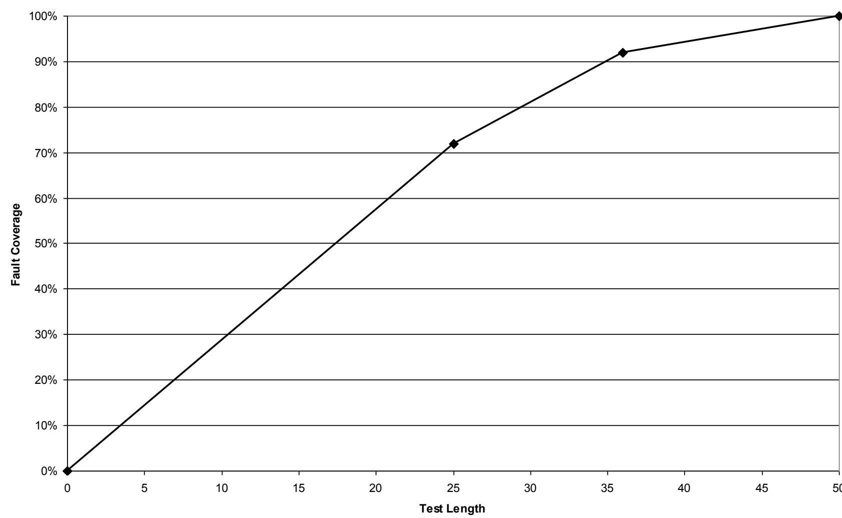

Figure D.9. Fault coverage over test length at TN4541

$$TT_{4541} = \frac{50 \cdot 261{,}56\text{kB} \cdot \frac{1024 \cdot 8}{32\text{kB}}}{180\text{MHz}} = 18.6\text{ms} \qquad (52)$$