Titre: Title:	Built-in self-test and self-repair architecture for defect-tolerant word-oriented large capacity memories
Auteur: Author:	Nader Ghattas
Date:	2004
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Ghattas, N. (2004). Built-in self-test and self-repair architecture for defect-tolerant word-oriented large capacity memories [Master's thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/7386/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/7386/
Directeurs de recherche: Advisors:	Yvon Savaria
Programme: Program:	Unspecified

UNIVERSITÉ DE MONTRÉAL

BUILT-IN SELF-TEST AND SELF-REPAIR ARCHITECTURE FOR DEFECT-TOLERANT WORD-ORIENTED LARGE CAPACITY MEMORIES

NADER GHATTAS

DÉPARTEMENT DE GÉNIE ÉLECTRIQUE

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION

DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES

(GÉNIE ÉLECTRIQUE)

DÉCEMBRE 2004

Your file Votre référence
ISBN: 0-494-01331-1
Our file Notre référence
ISBN: 0-494-01331-1

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

BUILT-IN SELF-TEST AND SELF-REPAIRARCHITECTURE FOR DEFECT-TOLERANT WORD-ORIENTED ULTRA-LARGE CAPACITY MEMORIES USING HIERARCHICAL REDUNDANCY

présenté par: GHATTAS Nader

en vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées

a été dûment acceptée par le jury d'examen constitué de:

M. KHOUAS Abdelhakim, Ph.D., président

M. SAVARIA Yvon, Ph.D., membre et directeur de recherche

M. BOIS Guy, Ph.D., membre

Acknowledgments

I would like to convey my sincere gratitude to all the people who helped me in the realization of this project. In particular, I would like to express my deepest appreciation to my supervisor professor Yvon Savaria. He provided me with guidance and support throughout my work on this thesis. His advices were always invaluable.

I would also like to thank the Canadian Microelectronics Corporation (CMC) for its continuous support and for permitting us the fabrication of the prototype chip through access to Taiwan Semiconductor's CMOS processes. It supplied us with all the required CAD tools, licenses and test equipment.

I am also grateful to Hyperchip Inc. for funding my research. It gave me the opportunity to interact with a number of technically brilliant research colleagues and I would like to heartily thank: Jacques Taillefer, Meng Lu, Bing Qiu and Daniel Picard.

Special thanks are due to all the members of "Groupe de Recherche en Microélectronique" (GRM) laboratory , particularly, to my brother, Hany Ghattas, for his continuous help and encouragement.

My parents deserve heartfelt thanks for their emotional support that kept me going. They were an inexhaustible source of encouragement throughout the period of my studies.

I extend an acknowledgment to our microelectronics technician, Martin Paré, for his help in the test procedure.

Last but not the least, I am grateful to our system administrator, Réjean Lepage, and computer technician, Alexandre Vesey, for their great effort in keeping the computer system constantly operational.

Abstract

Nowadays, SRAM chips use sub-micron technology to achieve high capacity memories. An increase in the memory capacity beyond the technology limits will normally result in very poor yields. The defect tolerance becomes then of a great importance for the realization of ultra-large capacity memory chips.

This project presents a self-testing and self-repairing strategy for ultra-high capacity memories. The self-testing and self-repairing structure applies tests that allow to locate faults and repair them without any external assistance from either test engineer or test equipment. This method will drastically improve the yield and reduce the production cost of embedded memories. The efficiency of self-testing and self-repairing strategy is supported by a hierarchical memory organization.

The redundant memory cells are introduced at different levels of hierarchy. At the lowest level of hierarchy, redundant words are introduced. If the local self-repairing logic can repair all the faults at the local level, the entire memory system can be restored to its fullest intended capacity. However, if a memory block has an excessive number of faults such that the local self-repairing logic is not able to restore its intended capacity at the local level, this memory block must be excluded from being accessed during normal

operations. Any attempt to access this faulty memory block will be diverted to a functional memory block.

A prototype memory chip of 4096 words by 4 bits with fault-tolerance has been designed and manufactured in CMOS 0.18 μm technology. Although the chip has a relatively modest storage capacity, it incorporates all the required self-testing and self-repairing structures. The memory array is divided into 4 blocks of 1024 words by 4 bits. Each block contains 2 redundant words. A redundant memory block is added at the top level.

A thorough study has been undertaken to identify the factors which can influence the area overhead, the yield and the access time of ultra-large capacity memories.

Résumé

De nos jours, les puces SRAM utilisent la technologie MOS sub-micronique pour réaliser des mémoires de haute capacité. Une augmentation de la capacité de la mémoire au-delà des limites de la technologie se traduira normalement par des rendements très faibles. La tolérance aux défectuosités devient alors d'une grande utilité pour la réalisation de puces de mémoire de grande capacité.

Ce projet présente une stratégie pour effectuer la réparation automatique des mémoires de très grande capacité. Cette structure automatique applique des tests qui permettent de localiser les défectuosités et de les réparer sans assistance externe d'un ingénieur ou d'un équipement de test. La méthode permet d'améliorer le rendement des mémoires et d'en réduire les coûts de production. L'efficacité de la structure automatique de test et de réparation des mémoires est supportée par une organisation hiérarchique.

Des cellules de mémoire redondantes sont introduites à plusieurs niveaux de la hiérarchie. Au plus bas niveau, des mots redondants sont introduits. Si la logique locale de réparation automatique peut réparer toutes les pannes au niveau local, le système de mémoire fonctionnera à pleine capacité. Cependant, si le bloc mémoire contient un nombre excessif de défectuosités qui ne peuvent être réparées automatiquement au niveau local, ce bloc ne doit pas être accessible durant le fonctionnement normal. Une tentative d'accéder à ce bloc doit être dirigée vers un bloc redondant fonctionnel.

Une puce de mémoire prototype de 4096 mots de 4 bits avec une tolérance aux pannes a été conçue et fabriquée en technologie CMOS 0.18 μm. Bien que la puce ait une capacité de mémoire relativement modeste, elle incorpore toutes les structures d'auto-test et de réparation automatique requises. La matrice de mémoire est divisée en 4 blocs de 1024 mots de 4 bits. Chaque bloc contient deux mots redondants. Un bloc redondant de mémoire est ajouté au niveau supérieur.

Une étude approfondie a été menée pour déceler les facteurs qui peuvent influencer le matériel additionnel, le rendement et le temps d'accès dans les mémoires de très grande capacité.

CONDENSÉ EN FRANÇAIS

INTRODUCTION

Il existe une limite à la complexité maximale des circuits réalisables avec les technologies d'intégration modernes. Au-delà de cette complexité, les rendements des procédés de fabrication chutent rapidement. L'emploi de redondance dans des structures tolérantes aux pannes permet d'atteindre une production plus économique des circuits de grande complexité.

Les circuits intégrés à grande échelle, tels que les processeurs vidéo, tendent à consacrer plus de transistors à la réalisation des mémoires embarquées qu'a leurs parties logiques. Il est prévu que la proportion de la complexité que représentent les mémoires embarquées augmentera encore plus dans le futur lorsque les puces réalisées deviendront des systèmes complets.

L'utilisation de RAM embarquées tolérantes aux pannes constitue une manière prometteuse pour préserver et augmenter le rendement global de production. La tolérance aux pannes des RAM peut être accomplie en utilisant des techniques de redondance combinées avec un auto-test intégré (BIST).

Une méthode efficace pour rendre des RAM statiques embarquées 'orientées-mot' tolérantes aux défectuosités fait l'objet de ce travail. La tolérance aux pannes de la RAM est basée sur une technique de redondance hiérarchique utilisant un auto-test déterministe ainsi qu'une logique d'auto-reconfiguration. Cette méthodologie est applicable à toute mémoire standard générée par un compilateur de mémoires. Il n'est donc pas nécessaire de connaître ou modifier la conception physique de la RAM.

Le prototype, présenté dans le cadre de ce mémoire, est basé sur une redondance à deux niveaux. Par conséquent, la mémoire est divisée en b blocs de RAM de même taille, auxquels s'ajoute un bloc redondant. Un circuit d'auto-test détecte toutes les pannes présentes dans chacun des blocs en les testant en parallèle. Chacun de ces $b+1$ blocs est équipé de deux mots redondants et il contient une logique simple d'auto-réparation pour masquer les pannes localement en remplaçant les mots défectueux. À haut niveau, les b blocs reconfigurés (sans défaut) forment la RAM.

Cette technique de redondance mène à une augmentation significative du rendement avec une surface additionnelle comparativement basse. En outre, seulement un délai modéré est ajouté sur le chemin emprunté par le signal, de telle sorte qu'il n'y ait presque aucune diminution dans la performance de la mémoire.

Ce condensé est structuré comme suit : la section 2 donne une vue d'ensemble sur le test des RAMs embarquées et une description de l'algorithme de test choisi ainsi que son

implantation. Dans la section 3, la logique de reconfiguration pour la redondance à deux niveaux sera présentée. Dans la section 4, une puce prototype qui a été conçue sera présentée et différentes analyses de rendement se rapportant à cette puce seront fournies. La section 5 offrira une brève conclusion et des recommandations pour des travaux futurs.

STRATÉGIES D'AUTO-TEST POUR MÉMOIRES EMBARQUÉES

L'utilisation de mémoires embarquées mène à des problèmes de test avec les méthodes de test externes conventionnelles, comme la mémoire n'est pas nécessairement connectée aux broches d'entrée et de sortie de la puce. En conséquence, diverses techniques d'auto-test intégré doivent être appliquées aux blocs de mémoire embarquée.

Le test des mémoires a fait l'objet de recherches depuis plusieurs années et divers algorithmes qui détectent une variété de classes de pannes ont été développés. Il existe deux méthodes générales pour tester les mémoires : la première est le test déterministe avec des algorithmes qui sont basés sur un modèle de pannes fixé précédemment et la seconde utilise des séquences de test pseudo-aléatoires. Cette dernière méthode ne se base sur aucun modèle de pannes spécifique et ne garantit pas de détecter une certaine classe de pannes avec une probabilité de 100%. Le test déterministe le plus répandu dans l'industrie est le "***March-test***". Ce dernier détecte toutes les pannes *collées-à*, *collées-ouvert*, de transition et de couplage.

L'auto-test intégré (BIST) a été retenu, puisqu'il permet de tester rapidement et de fournir des conditions de test réalistes et des durées de test plus courtes. Les étapes pour obtenir un algorithme de test efficace, utilisé dans la conception du BIST, sont les suivantes. En premier lieu, nous avons développé un modèle de pannes réaliste qui est basé sur les défectuosités réelles qui peuvent se produire dans la RAM pendant la fabrication. Ensuite, un algorithme de test a été identifié. Il s'agit du *13N March test*. Cet algorithme présenté au chapitre 2 donne un taux de couverture de 100% des pannes considérées. De plus, il conduit à une surface additionnelle et une dégradation de performance acceptables. Ces dernières sont dues au matériel supplémentaire d'auto-test requis.

Plusieurs blocs logiques sont nécessaires pour l'exécution de l'algorithme de test. Parmi les modules requis on retrouve:

- Un *compteur d'adresses* qui génère la séquence d'adresses de 0 à N-1.
- Un *générateur de données* qui génère la valeur du fond de données et son complément.
- Un *comparateur des données de test* qui se compose d'un comparateur au niveau du bit qui compare les données reçues du SRAM pendant une lecture aux données prévues.
- Un *contrôleur du BIST* qui exécute l'algorithme de test à l'aide d'un circuit séquentiel pour produire les signaux de commande nécessaires pour contrôler la RAM et les autres blocs du BIST.

TECHNIQUE DE REDONDANCE À DEUX NIVEAUX

Deux types de défectuosités peuvent se produire dans les blocs de mémoire, en l'occurrence, les petites défectuosités "*spot defects*" et les défectuosités de plus grande taille qui peuvent affecter plusieurs rangées de bits (ex. défectuosités de masque ou défectuosités groupés). Par conséquent, une technique de redondance à deux niveaux a été utilisée. Il est à noter qu'il est possible de réparer des défectuosités de grande taille avec une redondance à un seul niveau. Dans ce cas, la redondance se manifeste généralement sous la forme de multiple rangées redondantes dans la mémoire. Cependant, ceci conduit à une pénalité importante en terme de surface additionnelle. Ça peut aussi augmenter la complexité de la logique de reconfiguration et augmenter les délais de façon importante.

Notons que si la logique de reconfiguration est défectueuse, la mémoire entière sera probablement non-fonctionnelle. Cependant, ceci est peu probable, étant donné que la surface occupée par le circuit de réparation est minime en comparaison à celle de la mémoire.

Le partitionnement de la mémoire en plusieurs blocs, comme proposé dans ce travail, permet un test parallèle des blocs de mémoire. Si la taille nominale de la RAM est de 2^n x p bits, elle est divisée en $b = 2^{n-k}$ blocs, avec $(2^k + r)$ x p bits chacun, pour un décodage d'adresse commode, où r est le nombre de rangées redondantes par bloc de mémoire.

La reconfiguration à bas niveau implique plusieurs étapes. D'abord, il faut tester la mémoire, pour ensuite localiser toutes les pannes de façon à pouvoir finalement reconfigurer la mémoire en remplaçant les modules défectueux par des modules de remplacement. Ainsi, n'importe quelle rangée de la mémoire peut être remplacée par une des rangées redondantes.

Supposant que la mémoire contient un bloc redondant pour la reconfiguration à haut niveau, chacun des ***b+1*** blocs fournit un signal à la logique de contrôle de haut niveau indiquant son état de fonctionnalité. Si tous les ***b*** blocs réguliers sont sans défaut ou ont été reconfigurés avec succès, le bloc redondant n'est pas utilisé. Si un des blocs n'a pas pu être reconfiguré à bas niveau, un bloc redondant sera utilisé à sa place. Typiquement, ceci se produit si le nombre de modules de remplacement est trop petit ou s'il y a une défectuosité dans un élément critique qui n'est pas redondant. Avec plus qu'un bloc qui échoue, la mémoire ne peut pas être réparée. Elle fournit alors un signal au contrôleur indiquant qu'elle est défectueuse (résultat de test négatif).

Le contrôleur d'auto-réparation intégrée (BISR) implante l'algorithme de réparation pour remplacer les cellules défectueuses. D'abord, le BIST et la mémoire sont initialisés. Puis, l'algorithme de test est appliqué à la SRAM. Si une panne est détectée, le contrôleur de BISR vérifie que l'adresse actuelle n'avait pas été stockée précédemment. Dans un tel cas, elle assigne un espace libre dans les registres et enregistre la nouvelle adresse défectueuse. Au cas où le nombre d'échecs excéderait le nombre de mots redondants

dans le module de mémoire, le contrôleur remplacera ce module par un bloc redondant, s'il y en a un. Autrement, aucune réparation n'est possible.

Quand le test se termine, le système entre dans le mode d'opération normale. Dans ce mode, quand une adresse est consultée, elle est d'abord recherchée dans les registres d'adresse. Si elle correspond à une adresse trouvée défectueuse, l'adresse de remplacement du mot redondant est transmise. Autrement, l'adresse initiale est transférée directement au module de mémoire.

Le module de mémoire tolérante aux pannes se compose de plusieurs blocs :

- *Bloc de correction d'adresse* : ce bloc est responsable de réorienter une adresse entrante, destinée à un contenu défectueux dans la mémoire, vers une nouvelle adresse d'un mot redondant.
- *Bloc d'enregistrement des pannes* : ce bloc sauvegarde l'adresse de mémoire où la panne s'est produite.
- *Bloc de mémoire* : il s'agit de la matrice de mémoire SRAM synchrone et à double port, générée par le compilateur de *Virage Logic*.

Nous avons supposé que les petites défectuosités sont distribués uniformément et qu'il existe deux rangées redondantes par bloc de mémoire et un bloc redondant au plus haut niveau. Ainsi, cet arrangement peut tolérer jusqu'à ***2b*** petites défectuosités affectant une simple rangée et une défectuosité de grande taille pouvant couvrir la surface d'un bloc

entier de mémoire. Donc, avec une qualité suffisante du procédé de fabrication, le rendement de production peut atteindre presque 100%. Le processus de reconfiguration peut être répété à chaque mise sous tension, afin de détecter et de masquer les pannes, incluant celles qui apparaissent après le déploiement du circuit. Il n'y a donc aucun besoin d'appareil de contrôle externe pour exécuter l'auto-réparation.

IMPLANTATION ET RÉSULTATS

Une puce de mémoire prototype de 4096 mots par 4 bits comprenant toutes les fonctions discutées précédemment a été conçue et fabriquée en technologie CMOS 0.18 μm. Bien que la puce ait une capacité de mémoire relativement modeste, elle incorpore toutes les structures d'auto-test et de réparation automatique requises. La matrice de mémoire est divisée en 4 blocs de 1024 mots par 4 bits. Chaque bloc contient deux mots redondants. Un bloc redondant de mémoire est ajouté au niveau supérieur.

La conception entière a été réalisée en code VHDL. Les noyaux de mémoire ont été générés par le compilateur de mémoire de *Virage Logic* et ont été utilisés comme instances dans le code. Les simulations ont été exécutées sur Synopsys [2001]. Le dé occupe une surface de 4 mm^2 et est logé dans un boîtier PGA de 84 broches.

Des estimations du matériel additionnel requis pour l'auto-test et le circuit d'auto-réparation ont été effectuées. Ces estimations sont basées sur le dessin de masques réel

de la puce et la surface des cellules standards utilisées. En utilisant cette configuration de redondance, la surface supplémentaire diminue considérablement en divisant la RAM en plus de blocs. En outre, il peut être observé que la pénalité au niveau de la surface est assez grande pour les RAM de petites tailles. En conséquence, il n'est pas avantageux d'employer les stratégies d'auto-test et de réparation automatique avec les petites RAM. Il est préférable de les remplacer complètement.

En mettant en oeuvre ces stratégies d'auto-test et de réparation automatique, il n'est pas suffisant de considérer la taille de la RAM seulement. Sa configuration réelle (c.-à-d. le nombre de bits d'adresse et de données) doit être aussi prise en considération.

Une estimation du rendement est essentielle parce que trop de redondance signifie une perte dans la surface de silicium et trop peu mène à un faible rendement. Afin d'estimer l'amélioration du rendement grâce à la conception proposée et ainsi évaluer l'effet des différents paramètres du système, plusieurs calculs numériques ont été exécutés. Le rendement a été calculé en fonction de la densité de défectuosités en utilisant les équations basées sur les modèles de Poisson ainsi que le modèle binomial négatif.

L'amélioration du rendement, résultant de l'ajout de blocs redondants au niveau supérieur de mémoire, est généralement constatée à des basses densités de défectuosités. À des densités de défectuosités élevées, on observe que cette amélioration du rendement devient insignifiante. On remarque aussi que le rendement total augmente avec le nombre de

mots redondants inclus dans chaque bloc. Cependant, après une certaine valeur, il devient plus avantageux d'ajouter des blocs redondants au niveau supérieur que d'insérer plus de mots redondants localement.

Le rehaussement du rendement, dû à la division de la mémoire en de nombreux blocs plus petits, est plus significatif à des densités de défectuosités élevées. Cependant, avoir plus de blocs causera une augmentation de la surface de silicium, due à la complexité supplémentaire de l'adressage à haut niveau et au câblage additionnel requis. Ceci amène aussi une augmentation du temps d'accès de la mémoire.

CONCLUSION

Un nouveau concept de mémoire avec des capacités d'auto-test et d'auto-réparation intégrées, basé sur le remplacement des mots au lieu des rangées ou des colonnes redondantes, a été présenté. Il permet de réparer des RAM générées par un compilateur de mémoire sans modifier leur architecture de base. Cette méthodologie de test et de réparation de mémoire ajoute deux blocs principaux aux modules de mémoire. Une logique de BIST de mémoire met en oeuvre l'algorithme de test et examine la RAM lors de sa mise sous tension. De plus, un bloc de BISR stocke les adresses qui échouent pendant le test et remplace les mots défectueux. Le concept d'auto-test combiné avec la réparation automatique sera essentiel pour réduire les coûts et améliorer le rendement des futures puces de mémoire de très grande capacité.

TABLE OF CONTENTS

ACKNOWLEDGMENTS ..... **IV**

ABSTRACT ..... **VI**

RÉSUMÉ ..... **VIII**

CONDENSÉ EN FRANÇAIS ..... **X**

INTRODUCTION X

STRATÉGIES D'AUTO-TEST POUR MÉMOIRES EMBARQUÉES XII

TECHNIQUE DE REDONDANCE À DEUX NIVEAUX XIV

IMPLANTATION ET RÉSULTATS XVII

CONCLUSION XIX

TABLE OF CONTENTS ..... **XX**

LIST OF FIGURES ..... **XXV**

LIST OF TABLES ..... **XXVIII**

LIST OF ACRONYMS ..... **XXIX**

LIST OF APPENDICES ..... **XXXI**

1 INTRODUCTION AND MOTIVATION **1**

1.1 INTRODUCTION 1

1.2 SCOPE AND ORGANIZATION OF THE THESIS 5

2 TEST ALGORITHMS AND DETERMINING FACTORS IN CHOOSING A REPAIR ARCHITECTURE 11

2.1 INTRODUCTION 11

2.2 COMPARISON OF REPAIR METHODS 11

2.2.1 Conventional test and repair method 12

2.2.2 Advanced self-repairing method 13

2.3 TEST ALGORITHMS 15

2.3.1 Mscan test [15] 15

2.3.2 Checkerboard test [15] 16

2.3.3 Five-cell-neighborhood static-pattern-sensitive fault test [15] and [16] 17

2.3.4 Row/column weight-sensitive fault test [15] and [18] 18

2.3.5 Marching test [15, 44] 20

2.4 YIELD AND RELIABILITY FACTORS 22

2.4.1 Type and amount of redundant elements 22

2.4.2 Fault detection and localization 25

2.4.3 Redundancy allocation 25

2.4.4 Repair strategy 26

2.4.5 Reconfiguration mechanism 28

2.5 YIELD MODEL AND ESTIMATION 29

2.5.1 Yield with no redundancy 30

2.5.2 Yield using redundant rows or columns only 31

2.5.3 Yield using redundant rows or columns and redundant blocks 32

2.6 CONCLUSION 33

3 GLOBAL ARCHITECTURE AND THE SELF-TESTING FUNCTION 34

3.1 INTRODUCTION 34

3.2 GLOBAL ARCHITECTURE 34

3.2.1 Global Controller 37

3.2.2 Boundary Scan (JTAG) 38

3.3 THE SELF-TESTING FUNCTION 40

3.3.1 SRAM fault model 41

3.3.2 Test algorithm 44

3.3.3 Memory Fault Coverage 47

3.3.4 Architecture of the BIST 49

3.4 CONCLUSION 54

4 THE SELF-REPAIRING FUNCTION AND THE SRAM BLOCK 55

4.1 INTRODUCTION 55

4.2 THE SELF-REPAIR CONFIGURATION 55

4.3 METHODOLOGY 57

4.4 REPAIR ALGORITHM FLOW 59

4.5 FAULT-TOLERANT SRAM BLOCK ARCHITECTURE 60

4.5.1 Fault Registration Block 61

4.5.2 Address Correction Block 64

4.5.3 SRAM block 66
4.6 FAULT REPAIRING PROCEDURE 69
4.6.1 Test mode 69
4.6.2 Normal operation mode 72
4.7 CONCLUSION 73
5 IMPLEMENTATION AND RESULTS 74
5.1 INTRODUCTION 74
5.2 IMPLEMENTATION 74
5.3 AREA OVERHEAD ESTIMATIONS 76
5.3.1 Area Overhead Dependence with the RAM Block Size and the Number of RAM Blocks 78
5.3.2 Area Overhead in Dependence of the RAM Block Configuration 79
5.3.3 Area Overhead Projection for Ultra-Large Capacity Memory Chips 80
5.4 YIELD ANALYSIS 82
5.4.1 Redundant Blocks Effect 82
5.4.2 Redundant Words Effect 83
5.4.3 Block Size Effect 85
5.5 TIMING PENALTY 87
5.6 CONCLUSION 89
CONCLUSION AND FUTURE WORK 90
FUTURE WORK 92

REFERENCES **95**

APPENDIX A **103**

APPENDIX B **104**

LIST OF FIGURES

Figure 1.1 According to Semiconductor Industry Association (SIA) and International Technology Roadmap for Semiconductors (ITRS) 2001 [54], embedded memory will continue to dominate SoC content in the next several years, approaching 94% of the die area by 2014. 2

Figure 1.2 Memory yield improvement [54] 4

Figure 1.3 Simplified representation of the proposed concept. 6

Figure 1.4 Conceptual architecture of the proposed self -repairing SRAM. 8

Figure 1.5 The hierarchical redundancy structure. 9

Figure 2.1 External test and repair method 12

Figure 2.2 Advanced self-repairing method 14

Figure 2.3 Tiling a memory array for the checkerboard test : (a) pattern one: (b) pattern two 16

Figure 2.4 Tiling a memory array for the static-pattern-sensitive fault test: (a) phase 1; (b) phase 2 18

Figure 2.5 Partitioning a memory array into four in the row/column weight-sensitive fault test: (a) memory array with the border cells tested; (b) memory array partitioned into four. 19

Figure 2.6 Embedded memory repair strategies : (a) hard, (b) soft, (c) combinational, and (d) cumulative. 27

Figure 2.7 Memory with redundancy model 29

Figure 3.1 Global architecture of the implemented chip 36

Figure 3.2 Functional model of the global controller. 37

Figure 3.3 Basic architecture of the JTAG Boundary Scan. 39

Figure 3.4 BIST implementation procedure 41

Figure 3.5 Functional model of an SRAM chip 42

Figure 3.6 State diagram for two random cells in the memory array 49

Figure 3.7 Global architecture of the self-testable SRAM. 50

Figure 3.8 State diagram of the BIST controller 53

Figure 4.1 BISR basic architecture 58

Figure 4.2 The BISR algorithm 60

Figure 4.3 Fault-Tolerant SRAM Block. 61

Figure 4.4 The Fault Registration Block 62

Figure 4.5 Address Saving Block 62

Figure 4.6 Flow for storing a faulty address 64

Figure 4.7 Flow of address correction. 65

Figure 4.8 Symbol of the used memory 69

Figure 5.1 Layout view of the prototype chip 75

Figure 5.2 Area overhead for implementing the self-test and self-repair structure in dependence of the RAM block size for different numbers of blocks 79

Figure 5.3 Overhead in dependence of the total number of address bits and data bits for different RAM sizes divided into sixteen blocks plus one redundant block 80

Figure 5.4 Yield as a function of the defect density for different numbers of redundant blocks. 83

Figure 5.5 Yield as a function of the defect density for different numbers of redundant words per block and redundant blocks 84

Figure 5.6 Yield as a function of the defect density for different blocks sizes 85

Figure 5.7 Yield improvement due to dividing the memory into smaller blocks and its corresponding area overhead. 86

Figure 5.8 Self-repairing circuit on the memory critical path. 87

Figure 5.9 Waveform simulation showing the signal delay through the address correction block 88

Figure 5.10 Waveform simulation showing the signal delay caused by the top level addressing block 89

Figure 6.1 Word redundancy scheme with no flexibility. 92

Figure 6.2 Suggested word redundancy scheme with flexibility. 93

Figure 6.3 Sketch of the conceptual architecture using a CAM. 94

LIST OF TABLES

Table 3.1 The 13N marching test algorithm for SRAMs plus data retention test. 45

Table 4.1 Description of the input and output pins of the SRAM block 70

Table 5.1 Sizes of several functional blocks of the fault-tolerant memory prototype chip 76

Table 5.2 Estimation of the area overhead for different memory capacities. 81

Table 5.3 Different arrangements for a 128-Mb RAM. 86

LIST OF ACRONYMS

ASAP	Area, Speed, and Power
BIRA	Built-In Redundancy Allocation
BISR	Built-In Self-Repair
BIST	Built-In Self-Test
BISD	Built-In Self-Diagnosis
CAD	Computer Aided Design
CAM	Content Access Memory
CMC	Canadian Microelectronics Corporation
CMOS	Complementary Metal Oxide Semiconductor
DRC	Design Rule Check
EDA	Electronic Design Automation
FIFO	First In First Out
FSM	Finite State Machine
GRM	Groupe de Recherche en Microélectronique
HD	High Density
HS	High Speed
IC	Integrated Circuit
ITRS	International Technology Roadmap for Semiconductors
JTAG	Joint Test Action Group

LVS	Layout Versus Schematic
MCM	Multichip Module
PCB	Printed Circuit Board
PDA	Personal Digital Assistant
PGA	Pin Grid Array
RAM	Random Access Memory
RTL	Register Transfer Level
SIA	Semiconductor Industry Association
SoC	System on Chip
SRAM	Synchronous Random Access Memory
TAP	Test Access Port
TCK	Test Clock
TDI	Test Data In
TDO	Test Data Out
TMS	Test Mode Select
TRST	Test Reset
TSMC	Taiwan Semiconductor Manufacturing Company
ULP	Ultra-Low Power
VHDL	VHSIC (Very High Speed Integrated Circuits) Hardware Description Language
Y	Yield

LIST OF APPENDICES

APPENDIX A COMPILER PARAMETER RANGES 103

APPENDIX B MEMORY BLOCK CHARACTERISTICS 104

Chapter 1

INTRODUCTION AND MOTIVATION

In this chapter, the studied problem and the main objectives are introduced. Then, the strategy employed to obtain the proposed fault-tolerant SRAM and the thesis outline are presented.

1.1 INTRODUCTION

Today's ICs are memory dominated. Demand for more and more memory is driven by the need to support various high-bandwidth, high-capacity applications. This includes communications and networking products that must handle large amount of data in real time, applications for low-power portable devices such as PDAs and wireless phones, and high-speed processing applications requiring large and fast memories for program and data storage. In recent years, the amount of memory as a percentage of the total die area has increased dramatically attaining 70% or more. As illustrated in Figure 1.1, according to Semiconductor Industry Association (SIA) and International Technology Roadmap for Semiconductors (ITRS) 2001, embedded memories will continue to dominate SoCs (systems on chip) content in the next several years, approaching 94% of the die area by 2014.

Many designers have turned to embedded memories to improve system speed, lower power consumption and reduce costs. Thus, memories keep growing in size, which obviously affects the overall die size, requiring more silicon and increasing the production costs. Perhaps less obvious, but equally costly, is the increased risk of defects.

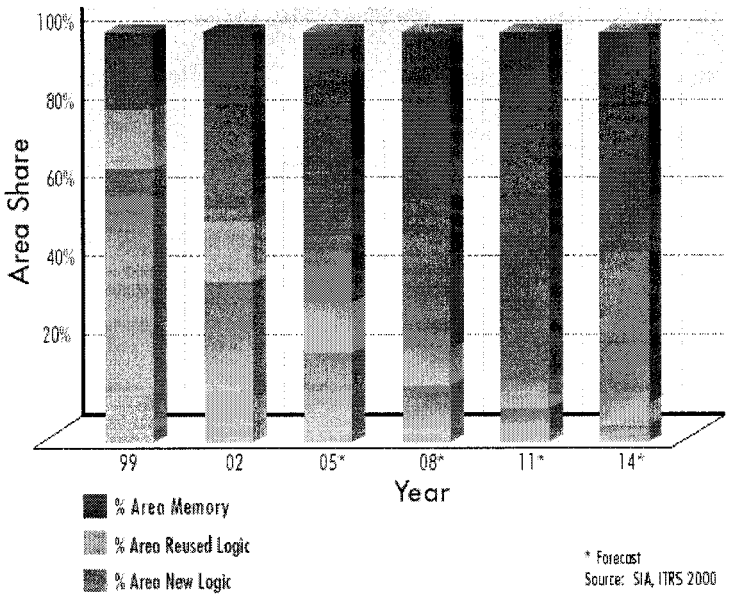


Figure 1.1 According to Semiconductor Industry Association (SIA) and International Technology Roadmap for Semiconductors (ITRS) 2001 [54], embedded memory will continue to dominate SoC content in the next several years, approaching 94% of the die area by 2014.

Having more embedded memory on a die makes it more vulnerable to failure due to manufacturing defects. Moreover, memory blocks are more susceptible to defects than logic. Memories are the densest circuits fabricated today. Because their transistors and

wires are packed so closely together, memory blocks suffer from a very high average number of physical defects per unit of area compared with other types of circuits. So, the greater the area of the die occupied by memory, the greater the risk of producing an IC that does not work and will cause the entire system to fail. Therefore, the total chip yield drops dramatically. One way to solve this problem is to enhance memories by adding redundant memory locations and test and repair algorithms [8] and [31]. Figure 1.2, presented by Zorian [54], compares the estimated yield of an embedded memory making use of these enhancements to the yield of the same memory, evaluated before adding test and repair capabilities, as a function of aggregate memory bit count in a SoC. It also shows the relationship between the memory yield and the die area occupied by memory. For instance, the yield of 24 Mbits of embedded memory, occupying 65% of the die area, can be increased from close to 20% to higher than 90% using an adequate repair strategy. This example assumes a 12-mm x 12-mm chip, in 0.13-micron technology, with a 0.12-per-square-centimeter memory defect density.

Recently, redundancy and repair have been extensively practiced for enhancing defect and fault tolerance. Unlike in legacy PCB (printed circuit board) or MCM (multichip module) based systems, embedded core components cannot be physically replaced once they are fabricated in a SoC. To realize enhanced manufacturing yield, a BISR (built-in-self-repair) must be used to allocate redundancy for embedded memory system cores. Building in redundancy allows the memories to be repaired. This has a significant impact on yield [22]. By being able to repair defects in the memory, the yield can be raised at an

acceptable level. For embedded memories totaling more than 1 MB, redundancy is essential to maintaining high yield.

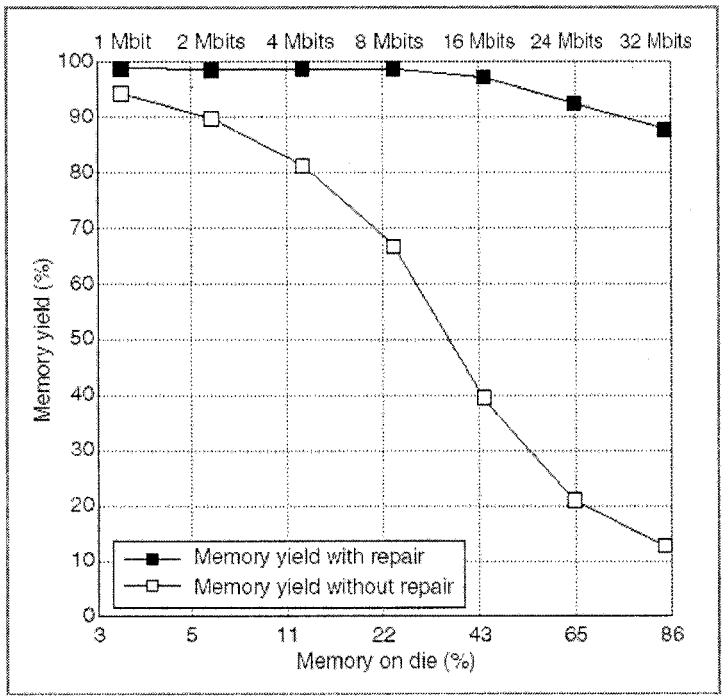


Figure 1.2 Memory yield improvement [54]

Therefore, fault-tolerance is an important issue in designing memory chips with ultra-large capacity. The common approaches to fault-tolerance include introducing spare rows and columns into the memory array [30]. The introduction of redundant rows and columns are often carried out by laser or by on-chip fuse/antifuse devices. These repairing techniques are referred to as "surgical repairing". They are simple techniques operated in a production environment; and they usually have minimum impact on memory performance. These techniques allow one-time programming and are

considered as hard repair. Any field related error correction and fault-tolerance cannot be achieved. Therefore, there is a growing interest to develop techniques for repairing these faults on the fly in any operating environment with the need of no external equipment.

1.2 SCOPE AND ORGANIZATION OF THE THESIS

This project goal was to develop a built-in self-testing (BIST) and self-repairing (BISR) SRAM offering the following characteristics:

- Self-testing function capable of detecting faulty cells that need to be repaired.
- Self-repairing function performed on-line and transparently, without the user intervention.
- Self-repairing independent from the memory physical implementation. Most of the self-repairing approaches proposed so far are based on row/column repair strategy requiring the knowledge of the memory layout. To achieve user level memory repairing, a method based on cell replacement is presented.

Figure 1.3 shows a simplified representation of the proposed concept. A BIST tests the memory and provides information about faulty cells, while a BISR uses this information to repair the memory [10].

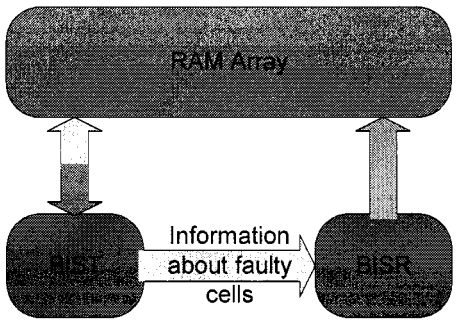


Figure 1.3 Simplified representation of the proposed concept.

The proposed word oriented static RAM Built-In-Self-Repair methodology allows to use standard RAMs generated from memory compilers without spare rows or spare columns and add to them some redundant logic. This permits to build a self-repairing memory without modifying the memory module. The redundancy and BIST logic is fully parametric, synthesizable and reusable. It is highly flexible and can be used with various memory types that do not have any redundancy capabilities. The implemented self-repair strategy using redundant words can be summarized as follows:

1. Partition a large SRAM array into several identical sub-blocks. This is generally a common practice for high performance embedded RAMs [2], [3] and [11]. Each sub-block is supported by dedicated logic to allow replacing defective cells [12].
2. Repair each sub-block independently of the others. In other words, each sub-block has its own spare words that are not shared across sub-blocks.

3. Provide 2 spare words for each sub-block. When a repair is needed, the entire word is used to replace an equivalent defective word in the SRAM.

4. Provide one spare sub-block at the top level to replace a sub-block that has more than 2 different defective words and is therefore locally unrepairable.

5. Develop and implement an on-line repair algorithm [4], [7], [23] and [34] that detects and diagnoses the failure information on the fly during self-test and maintains an up-to-date fault map which consists of a list of the locations where a failure occurred and the corresponding spares used to replace them.

As the memory, the decoder and the read-write logic form an integrated full custom module generated by a RAM generator, the self-test as well as the self-configuration strategy must not intervene between the memory field, the decoder and the read-write logic.

The concept of enhancing the SRAM with self-repairing capability is illustrated in Figure 1.4. First, a suitable test algorithm is chosen and implemented as a built-in self-test (BIST) with low area overhead. A multiplexer placed in front of the SRAM is widely used to provide the memory with the test pattern generated by the BIST. Then, a circuit that stores the faulty addresses and redirects them into functional ones is developed. When a fault is detected, the self-repair logic automatically reconfigures the memory

using redundant cells in the form of words and blocks replacing the defective ones [33]. When a faulty cell is addressed, the circuit reacts by changing the address value to the address of the redundant cell.

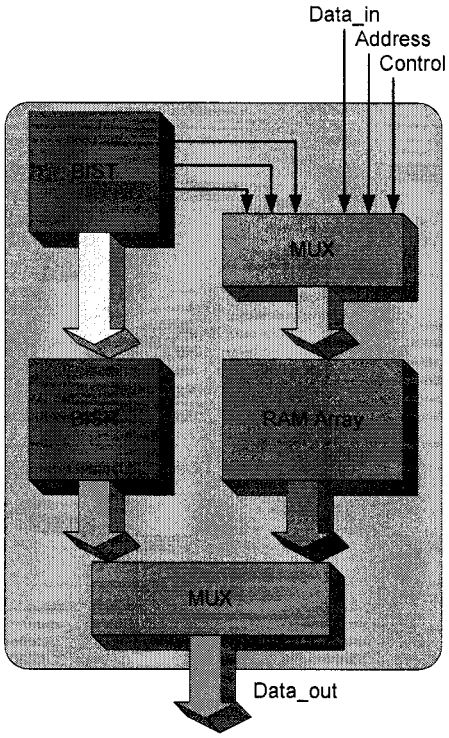


Figure 1.4 Conceptual architecture of the proposed self-repairing SRAM.

In this approach, a two-level redundancy has been implemented [32], as illustrated in Figure 1.5. Therefore, the RAM is split up into B blocks of M words by N bits. On the lower level, each block is equipped with additional memory cells in the form of R spare words. On the higher level, S additional redundant blocks are provided to mask larger defects. This hierarchical redundancy structure achieves a significantly higher yield with a rather low area overhead [20].

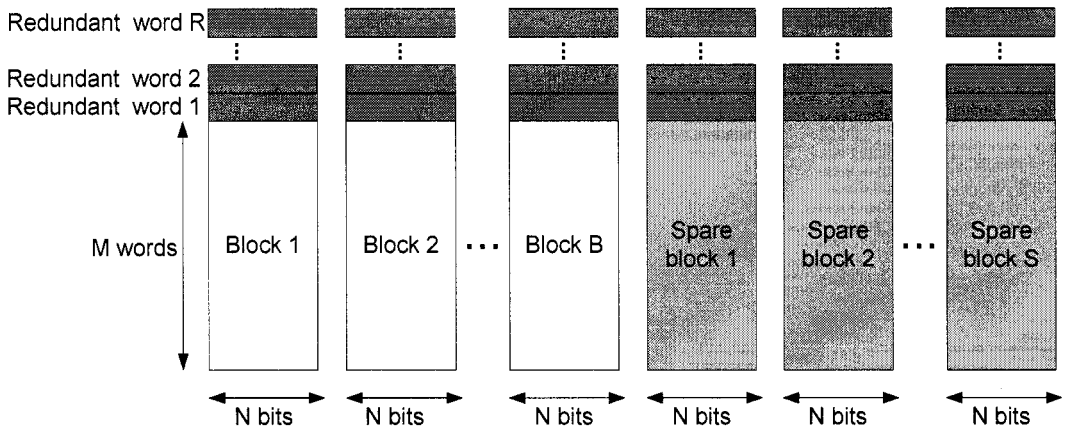


Figure 1.5 The hierarchical redundancy structure.

The thesis is organized as follows. In chapter 2, all the determining factors that have to be considered when selecting a self-repairing technique will be treated. These factors influence the repair efficiency and thus the yield of the embedded memory. A model will be proposed to estimate the overall yield considering different redundancy configurations. In chapter 3, the global architecture of the proposed self-testing and self-repairing SRAM design will be introduced. This architecture is divided into two major segments: the self-testing function which makes the object of chapter 3 and the self-repairing component which will be treated in chapter 4. In chapter 5, the implementation procedure of the entire system will be given as well as the most relevant results and analysis. Finally, a brief conclusion and some suggestions for future work will be presented in chapter 6.

In this chapter, the yield deterioration problem resulting from including more embedded memory on the die has been exposed. Existing solutions to deal with this problem have been considered and the major objectives have been specified. Finally, the methodology used to realize a self-testing and self-repairing embedded memory has been proposed.

Chapter 2

TEST ALGORITHMS AND DETERMINING FACTORS IN CHOOSING A REPAIR ARCHITECTURE

2.1 INTRODUCTION

In this chapter, the traditional test and repair method is compared to an advanced self-repairing technique. Several test algorithms found in the literature are enumerated with a brief description. The major factors that influence the yield and reliability of the embedded memory are enumerated. A yield model that can be applied to predict the yield of a memory array using different redundancy configurations is also developed in this chapter.

2.2 COMPARISON OF REPAIR METHODS

The conventional method of using external test and repair equipments presents several drawbacks. To overcome them, a more advanced method to repair embedded memories is proposed.

2.2.1 Conventional test and repair method

The most common approach to perform memory repair is using external test and repair. As illustrated in Figure 2.1, this method can be divided into four main steps, all taking place at wafer level:

1. An external memory tester applies a wafer-level memory test on the embedded memory with redundancy and stores the failed bit map in a large capture memory. Then this failed bit map is used by a redundancy allocation software to determine the best way to allocate redundant resources to replace defective locations and generate the reconfiguration data.

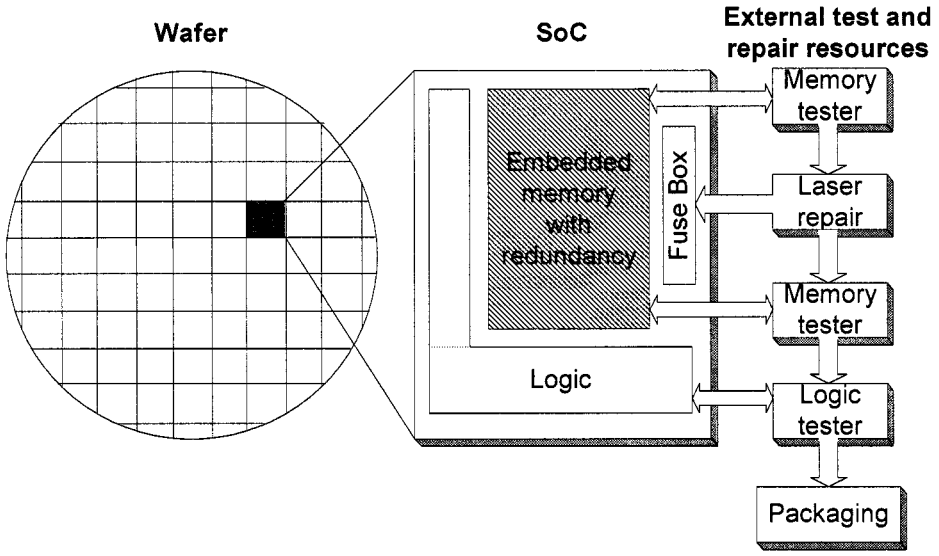


Figure 2.1 External test and repair method

2. These reconfiguration data are then transferred to the laser repair equipment, which programs them by blowing fuses in laser fuse boxes corresponding to the defective memories.

3. When the repair is accomplished, a retest is applied by a second memory tester to make sure that the repair was successfully performed.

4. Before packaging, an external logic tester applies a test on the remaining logic part of the SoC.

Due to the massive use of external and expensive equipment, this test and repair method largely increases the total manufacturing cost of a chip [38]. In addition, this method requires a high bandwidth access to the embedded memory during memory test performed by the functional tester. The fact that this method uses a general-purpose redundancy allocation algorithm restricts its repair efficiency. Furthermore, this technique does not allow any field repair.

2.2.2 Advanced self-repairing method

To overcome the drawbacks of the conventional external test and repair method, presented in the preceding section, an effective self-repairing technique using an on-chip logic structure, which can be activated in any operating environment, is highly desirable [6]. This will allow a periodic field level repair and power-up soft repair for the embedded memories. The process of self-repairing, shown in Figure 2.2, can be divided into several steps, all built-in the RAM architecture:

1. Built-In Self-Test (BIST): In the first place, a test algorithm is executed on the memory array to detect faults.

2. Built-In Self-Diagnosis (BISD): If a fault is detected, it is necessary to locate it (diagnosis).

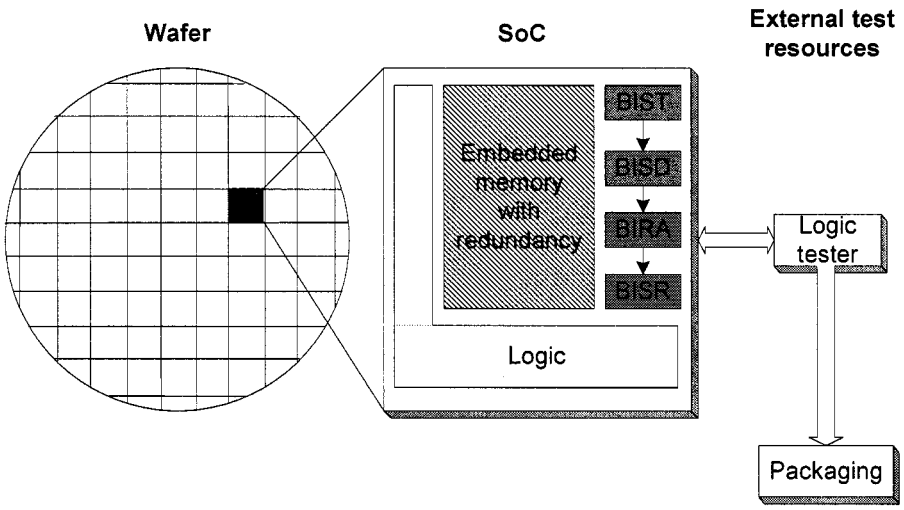


Figure 2.2 Advanced self-repairing method

3. Built-In Redundancy Allocation (BIRA): Here, it allocates the redundant memory space available.

4. Built-In Self-Repair (BISR): Finally, it replaces the faulty cells.

This method reduces considerably the manufacturing cost, since external test and repair equipment is not needed as discussed in [8] and [39]. The logic tester by itself is sufficient to perform the memory test and repair process followed by testing of the random logic blocks. Moreover, it improves the repair efficiency due to the fact that the memory can be reconfigured several times in the field.

2.3 TEST ALGORITHMS

Over the years, several algorithms of different complexities ranging from O(N) to $O(N^2)$, where N is the total number of memory cells, have been developed to test RAMs. The early algorithms were developed in an ad hoc manner; whereas the later algorithms were specifically designed to detect faults associated with various fault models. Many of those algorithms have been applied to BIST. Three classes of test algorithms can be distinguished: deterministic, pseudo random and pseudo exhaustive tests [55]. A review of these test algorithms can be found in [16], [17], [43] and [47].

All test algorithms consist of a sequence of writes and reads applied to the cells in the memory array. In the subsequent subsections, $W_i \leftarrow v$ means writing value v into cell i and R_i (= v) stands for reading cell i, with v as the expected value.

2.3.1 Mscan test [15]

Memory scan is a trivial test procedure developed in an ad hoc manner. It consists of writing a 0 to each cell then reading it, followed by writing a 1 and reading it. The formal algorithm can be summarized as shown below:

For i = 0, 1, …, N – 1

$W_i \leftarrow 0$

R_i (= 0)

$W_i \leftarrow 1$

R_i (= 1)

The deterministic fault coverage of this test method is rather low. All that is known at the end is that there is at least one cell in the RAM that can be set to 0 and 1. This is because a fault in the decoder may cause the same cell to be referenced each time. Its length is 4N since it performs four operations on each cell.

2.3.2 Checkerboard test [15]

This simple algorithm, developed in an ad hoc manner, is designed for two-dimensional memory architectures. The algorithm fills the memory array with a checkerboard pattern by writing 0's and 1's in alternate cells. The two patterns shown in Figure 2.3 are written. The cells are read after the application of each checkerboard pattern.

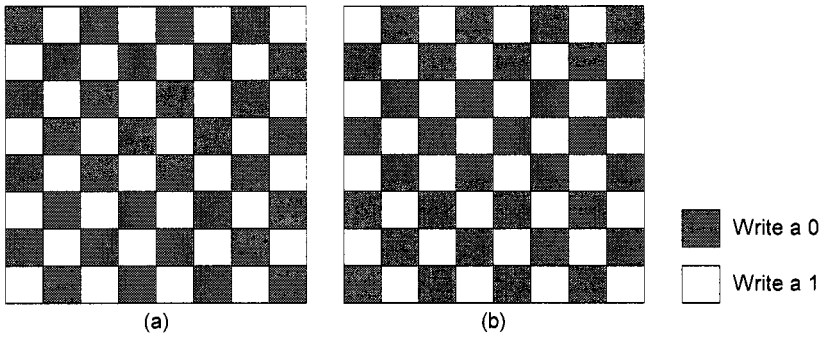


Figure 2.3 Tiling a memory array for the checkerboard test : (a) pattern one: (b) pattern two.

The formal algorithm is given below:

Step 1. $W_{(i,j)} \leftarrow 0$ for i + j = even

$W_{(i,j)} \leftarrow 1$ for i + j = odd

Step 2. $R_{(i,j)}$ (= 0) for i + j = even

$R_{(i,j)}$ (= 1) for i + j = odd

Step 3. Repeat steps 1 and 2, interchanging 0's and 1's.

The deterministic fault coverage of this test procedure is rather low. As with the Mscan test, a decoder fault may cause only four cells at most to be referenced. Therefore, all that is known at the end of this test is that at least four cells in the RAM can be set to 0 and 1.

2.3.3 Five-cell-neighborhood static-pattern-sensitive fault test [15] and [16]

Many algorithms have been proposed to detect five-cell-neighborhood pattern-sensitive faults. All these algorithms are based on tiling the memory array. Kinoshita and Saluja [25] suggested an algorithm based on the tiling arrangements shown in Figure 2.4. The unmarked cells are the base cells. Each base cell is surrounded by four characters (A, B, C and D).

The first phase of the test uses the tiling arrangement illustrated in Figure 2.4a. During this phase, the base cells are kept fixed at logic 0. The five-cell-neighborhood patterns are applied to the base cells using all four-tuples (16 patterns), consisting of Boolean

variables A, B, C, and D. The base cells are read after the application of each pattern. The second phase uses the tiling arrangement of Figure 2.4b, and the above process is repeated. Then both phases are repeated with the base cells at logic 1. This algorithm is based on the abstract logical neighborhood, as opposed to the actual physical neighborhood, assuming that the two neighborhoods are identical. If the assumption is invalid, such tiling methods can go haywire. Franklin and Saluja proposed in [18] a test algorithm to detect five-cell physical neighborhood pattern sensitive faults, even if the logical and physical addresses are different and the physical-to-logical address mapping is not available. This algorithm has a test length of $O(N[\log_3 N]^4)$.

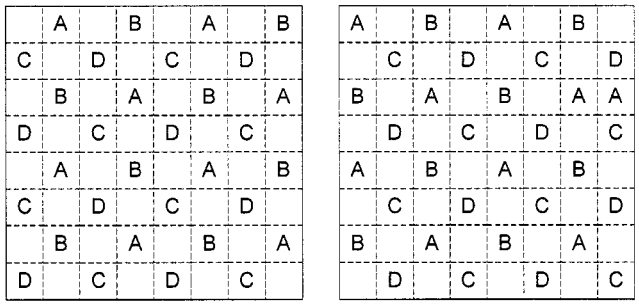


Figure 2.4 Tiling a memory array for the static-pattern-sensitive fault test: (a) phase 1; (b) phase 2

2.3.4 Row/column weight-sensitive fault test [15] and [18]

Different test algorithms of varying test length have been proposed for testing RAMs for row/column weight-sensitive faults [19]. All the tests are of length $O(N^{3/2})$ and use divide and conquer by recursive partitioning as the basic strategy. First, the border cells of an array are tested. Then, the two middle rows and columns are tested, thereby

effectively partitioning the array into four, as illustrated in Figure 2.5. Partitioning continues recursively until all the cells of the array are tested. The test procedure can be conceptually divided into the following steps:

Step 1 Test the four corner cells of the RAM simultaneously, for states with cell values 0 and 1.

Step 2 Test the border cells consisting of the top row, bottom row, leftmost column and rightmost column cells for states with cell value 0.

Step 3 Test the partitioning cells consisting of the two middle rows and middle columns, for states with cell value 0. This step partitions the array into four parts, with the border of each partition completely tested for states with cell value 0.

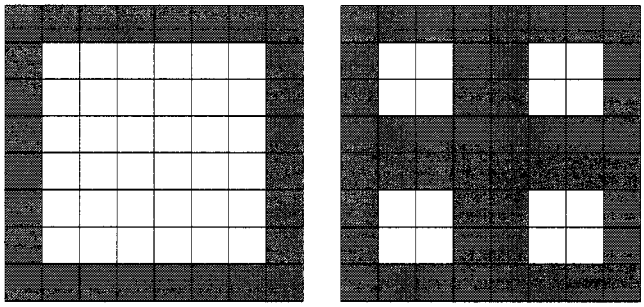


Figure 2.5 Partitioning a memory array into four in the row/column weight-sensitive fault test: (a) memory array with the border cells tested; (b) memory array partitioned into four.

Step 4 Partition the memory array recursively by executing step 3 till all the cells within a partition are tested completely. At the end of this step, all cells in the array will be completely tested for cell value 0.

Step 5 Repeat steps 2 to 4 for testing the non-corner cells of the array for states with cell value 1.

Testing the border cells involves scanning the memory array as in the marching test, and this helps to detect decoder faults, stuck-at, stuck-open, transition and coupling faults. However, the row/column weight-sensitive fault test has a higher length.

2.3.5 Marching test [15, 44]

The most widely used deterministic test algorithm in the industry is the marching test. The reason for its popularity is its simplicity coupled with a moderate fault coverage. This algorithm detects all faults affecting the decoder, all stuck-at faults in the read/write logic and all single and multiple stuck-at, stuck-open, transition and non-inverting coupling faults.

The March algorithm consists of a finite sequence of March elements applied to each cell in the memory in a given order [44]. First, it initializes the memory array to all 0's, and then scans the memory cells in ascending and descending orders. For each cell, scanning involves reading the cell for the expected value, writing the complement value, and reading it again. The formal algorithm can be expressed as follows:

Step 1. For $i = 0, 1, \ldots, N - 1$

$W_i \leftarrow 0$

Step 2. For i = 0, 1, …, N – 1

Ri (= 0)

$W_i \leftarrow 1$

Ri (= 1)

Step 3. For i = N-1, N-2, …, 0

Ri (= 1)

$W_i \leftarrow 0$

Ri (= 0)

Step 4. Repeat steps 1 through 3, interchanging 0's and 1's.

The idea behind this algorithm is that, while scanning the memory in ascending order, any direct coupling between the current cell and a higher address cell is detected when reading the latter. Moreover, any error in the higher address cell due to decoder faults will also be detected. Similarly, scanning the memory in the descending order detects all the effects on lower address cells.

Different variations of the marching test, all of complexity O(N), have been seen in the literature [15], [46] and [49]. In [14], Dekker suggested a 9N and a 13N algorithms for testing SRAM's. According to him, these test algorithms count among the best of all available test algorithms as far as fault coverage and testing time are concerned. The 13N

test algorithm has been chosen to implement the BIST in our design and will be described in more details in chapter 3.

2.4 YIELD AND RELIABILITY FACTORS

According to Zorian [53], several factors can contribute to the yield and reliability optimization of an embedded memory, namely: the type and amount of redundant elements, the fault detection and localization algorithm, the redundancy allocation algorithm, the repair strategy, and the reconfiguration mechanism. The way each factor is selected plays a significant role in the repair efficiency and thereby in the yield of the embedded memory. Each one of these factors will be treated in this section.

2.4.1 Type and amount of redundant elements

The type and amount of redundant elements incorporated in the memory constitutes a determining factor in the yield optimization and the repair efficiency. A prediction of the post-repaired yield allows determining the amount of redundancy required to attain some target yield. Deciding on the amount of redundancy to include in the embedded memory is very critical because implementing too much redundancy will cause a waste in the silicon area, and implementing too little will result in a poor yield. Typically, memories smaller than 1 Mbit do not necessitate redundancy.

2.4.1.1 Types of redundancy

Different types of redundancy exist [5] and [37]. This section gives an overview of each type.

- ***Wordline (row) redundancy***

  Wordline redundancy allows replacing one or more rows with spare ones. A special dedicated logic is added around the memory array to assure this replacement. The only timing penalty of this configuration is a slight degradation of the address setup time, caused by the presence of an address comparator used to detect the access to the redundant location. More details concerning this timing penalty will be provided in section 5.5. Moreover, the area overhead to implement this type of redundancy is quite low.

- ***Bitline (column) redundancy***

  Bitline redundancy is similar to the wordline redundancy, except that spare columns are added to the embedded memory instead of rows. Designing an efficient architecture to implement bit line redundancy allowing the redundant columns to be shared throughout the memory array is usually difficult due to the presence of bit lines multiplexers which choose one bit line among several ones before the output sense amplifier.

- ***Word (cell) redundancy***

  Word redundancy consists in adding a few redundant words, each corresponding to a logical RAM address. This organization allows a more flexible configuration compared to physical wordlines [40].

2.4.1.2 Redundant space configuration

Embedded memories can use a single type of redundant elements or a combination of multiple types. Based on the three redundancy approaches described in section 1.4.1.1, redundant space can be inserted in three different ways in the memory array:

1. ***Row/Column Only***

   The memory is provided with spare rows or columns. In this case, all spares are of the same type, row or colums but not both. When a fault is detected, the row/column where the fault occurred is replaced by one of the spare rows/columns. However, this configuration does not use the redundant space in an optimal way, given that a whole row/column has to be allocated to repair a single fault.

2. ***Row-column***

   The memory is provided with both spare rows and spare columns. Any fault can be repaired by using either a spare row or a spare column. When multiple faults

occur, the best combination of spare rows/columns is chosen to optimize the redundant space while repairing the faults efficiently.

3. ***Word (cell) only***

   The memory is enhanced with spare words. Each word corresponds to an addressable location in the RAM. When a fault occurs, the faulty address is redirected to a spare word, instead of replacing a complete row or column. Therefore, this configuration allows an optimal use of the redundant space and it has been adopted in the proposed design.

2.4.2 Fault detection and localization

In order to improve the memory yield and reliability, faults need not only to be detected; but also to be located for further repair. Fault localization is required to identify the faulty cells that need to be replaced. Therefore, the test algorithm should employ efficient procedures for effective detection and location of the faulty cells to reach a higher yield.

2.4.3 Redundancy allocation

Redundancy allocation consists in determining the best way to assign redundant resources to replace faulty locations [28]. In the case where only one type of redundant elements is used, row-only redundancy for instance, the redundancy allocation is

straightforward. On the other hand, if a combination of various redundant elements types is used, an optimal reconfiguration has to be found and redundancy allocation becomes more complex. An algorithm needs to be developed and implemented to determine the most efficient solution to repair the memory [9]. This redundancy allocation algorithm has to run on the fly, meaning in real-time, while the fault detection and localization are performed.

2.4.4 Repair strategy

The repair strategy consists in the methodology employed to perform the actual repairing. The selection of a repair strategy can influence dramatically the repair efficiency and thereby the yield. Four different strategies, illustrated in Figure 2.6, can be distinguished. A brief description of each one of them is given below:

2.4.4.1 Hard repair

Hard repair necessitates the use of a permanent storage to retain the repair information after power is turned off. This repair is performed only once using some fuse box and does not need retesting or reconfiguration at every power-up.

2.4.4.2 Soft Repair

Soft repair is performed at every power-up for an unlimited number of times. It does not require a fuse box given that the repair information does not need to be held upon power-

down. This method is able to detect different faults at different times allowing field repair and thus improving reliability.

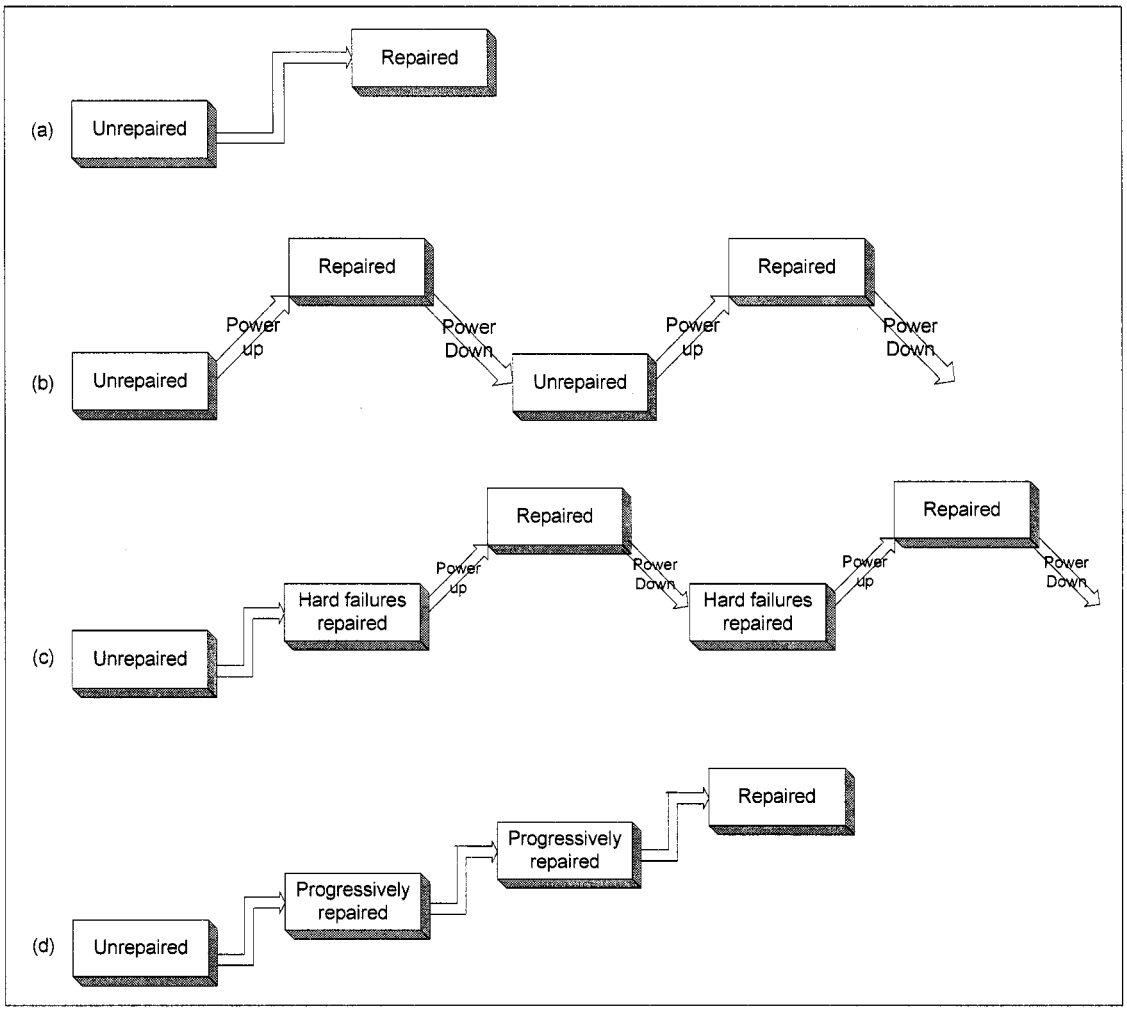


Figure 2.6 Embedded memory repair strategies : (a) hard, (b) soft, (c) combinational, and (d) cumulative.

2.4.4.3 Combinational repair

Combinational repair starts with a hard repair in the factory followed by a soft repair in the factory or in the field. This method offers the best of both hard and soft repair. It is

better than each one operating alone. However, the information acquired from each soft repair is not stored and cannot be reused upon the next power-up.

2.4.4.4 Cumulative repair

At the beginning of each test and repair cycle, the memory uses the previously acquired repair data and adds new data obtained during the current cycle by storing them in a reprogrammable nonvolatile fuse box [35]. In that way, it constructs progressively its repair information. This method can reach higher yields by identifying more faults over time. Programming the fuse box multiple times allows detecting defects under different environmental conditions, an unfeasible characteristic using the one-time repair. Therefore, the cumulative methodology achieves the highest repair efficiency.

2.4.5 Reconfiguration mechanism

The memory reconfiguration can be accomplished through true repair or a bypass technique. The bypass technique consists in replacing the faulty locations by redundant ones using multiplexers. On the other hand, true repair disconnects the faulty bit line from the power source while replacing it, thus preventing any possible memory performance degradation that can be caused by a leakage current. However, true repair requires full access to the memory layout since the redundancy multiplexers need to be integrated in it, compared to the bypass technique where the multiplexers are implemented at the RTL outside the RAM. This may have a slight impact on the memory performance and the area overhead.

2.5 YIELD MODEL AND ESTIMATION

It is not easy to predict with exactitude the yield of a memory array using redundancy. Nevertheless, assuming the defect distribution is exactly represented by the Poisson and the binomial distributions, the suggested model offers a close approximation of the actual yield. This expected value can be evaluated for many possible internal structures when designing a memory with redundancy. In the proposed fault tolerant array structure, shown in Figure 2.7, the SRAM memory chip is partitioned into B identical blocks of (M+R) rows by N bits. Each block contains R redundant rows for local repair within the same memory block. Moreover, the model includes S redundant blocks to replace the defective ones (i.e. blocks with more than R faulty rows).

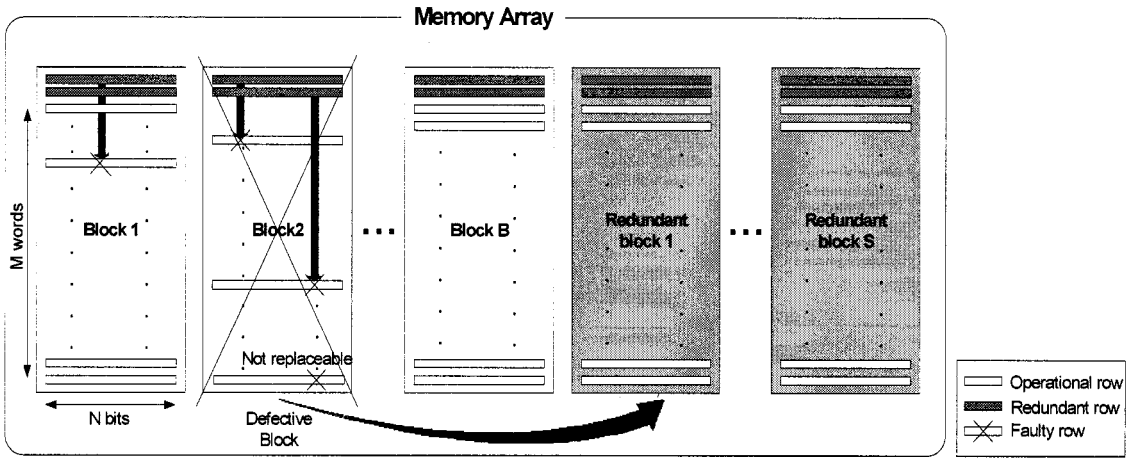


Figure 2.7 Memory with redundancy model

The yield analysis is carried out by applying two extensively used analytical models: the Poisson distribution and the binomial distribution [26], [27] and [36]. First, the Poisson distribution is employed to calculate the yield of a single bit-cell. Then, the negative

binomial distribution is used through several steps to finally obtain the yield of the memory core [41].

The yield of a single cell is the probability of one cell to operate properly and can be given by equation (2.1).

$$Y_{cell} = e^{-\lambda cell} = e^{-(D_o \cdot A_{cell})} \tag{2.1}$$

where λ_{cell} designates the average number of faults per cell, D_o is the defect density of the process and A_{cell} is the bit-cell area.

2.5.1 Yield with no redundancy

If no redundancy is used, the memory is unable to tolerate any faults. Any presence of faults cannot be repaired and results in a defective memory. The yield of such a memory is defined as the probability of not containing any faults. The yield of one memory block without redundancy, i.e. the probability that the block will operate with no faults, can be described by equation (2.2).

$$Y_{bnr} = e^{-(D_o \cdot A_{cell} \cdot M \cdot N)} \tag{2.2}$$

where M is the number of words, N is the number of bits per word in one block and Y_{bnr} stands for the yield of one block with no redundancy. Given that there are B blocks in the memory core and that they all have equal probability of being defective, the yield of the entire core can be expressed by equation (2.3).

$$Y_{nr} = (Y_{bnr})^{B} \tag{2.3}$$

where Y_{nr} stands for the core yield with no redundancy.

2.5.2 Yield using redundant rows or columns only

If only one type of redundant elements is used: rows or columns, the yield can be defined as the probability of having a number of faulty rows or columns that does not exceed the number of redundant elements incorporated in the memory block [24]. The case where the memory only has redundant rows has been taken as an example. Making use of the Poisson distribution, the yield of one row, which can be viewed as the probability that this row contains no faults, is given by equation (2.4).

$$Y_{row} = e^{-(D_o \cdot A_{cell} \cdot N)} \tag{2.4}$$

Using the binomial distribution, the probability of having r faulty rows out of the M rows present in a block is expressed in equation (2.5).

$$\text{P (r faulty rows)} = \binom{M}{r} (1 - Y_{row})^{r} Y_{row}^{(M-r)} \tag{2.5}$$

Assuming R_{row} is the number of redundant rows included locally in each block, this memory can tolerate a maximum of R_{row} faulty rows per block. Thus, the yield of one block, i.e. the probability of finding no more than R_{row} rows with defects, can be described by equation (2.6).

$$Y_{block} = \sum_{r=0}^{R_{row}} \binom{M}{r} (1 - Y_{row})^r Y_{row}^{(M-r)} \tag{2.6}$$

These are the probabilities for the rows that need to be fixed, and for which there are redundant rows available within the block. The yield of the entire core partitioned into B blocks can be given by equation (2.7).

$$Y_{rro} = (Y_{block})^B \tag{2.7}$$

where Y_{rro} stands for the yield of a memory core provided with redundant rows only.

2.5.3 Yield using redundant rows or columns and redundant blocks

Moreover, if the memory is enhanced with spare blocks to replace the defective ones, the yield of the whole core becomes as given in equation (2.8).

$$Y_{core} = \sum_{i=0}^{S} \binom{B}{i} (1 - Y_{block})^i Y_{block}^{B-i} \tag{2.8}$$

where S is the number of spare blocks in the core. This corresponds to the probability of having no more than S locally unrepairable blocks, i.e. blocks that contain more than R_{row} faulty rows.

Results obtained using the equations in sections 2.5.2 and 2.5.3 will be presented in section 5.4.

2.6 CONCLUSION

In this chapter, an advanced self-repairing strategy has been weighed against the traditional test and repair method. It reduces significantly the manufacturing cost by eliminating the need for external test and repair equipments. Also, it increases the repair efficiency by allowing field repair. Many test algorithms were discussed and compared. The 13N algorithm was chosen for implementation for its simplicity and its high fault coverage. The main factors affecting the yield of embedded memories have been discussed. Moreover, a model to compute the yield of a memory array using redundancy has been presented.

Chapter 3

GLOBAL ARCHITECTURE AND THE SELF-TESTING FUNCTION

3.1 INTRODUCTION

In this chapter, a global architecture of the proposed self-testing and self-repairing SRAM is presented. The procedure to obtain an efficient Built-In-Self-Test (BIST) is suggested. A fault model is adopted and a test algorithm is developed and implemented. The BIST architecture with all its sub-blocks is also treated here.

3.2 GLOBAL ARCHITECTURE

In telecommunication, First-In-First-Out (FIFO) memories are becoming widely used as buffers between subsystems operating at different data rates [45]. In the suggested architecture, we have one host that needs to communicate with 4 terminals. However, this host is sending data at a higher rate than the clients can receive. Therefore, a buffer storage between the host and the client is needed. This buffer is implemented using a dual-port SRAM-type FIFO with 2 independent counters to specify the read and write

addresses to the memory via separate ports [48]. Moreover, these SRAM's are enhanced with a self-testing and self-repairing capability.

As shown in Figure 3.1, at power on, the five SRAM blocks, including the redundant one, are tested simultaneously by the shared BIST. Given that the four memory blocks are identical, a unique BIST has been used for diagnosis in order to minimize the area overhead. However, a self-repairing scheme, including the test data comparator, was incorporated in each block to support memory reconfiguration in case of failure.

After the test is performed and the memory is reconfigured if necessary, the global signal "Test Enable" goes low, indicating that the entire self-testing and self-repairing process is completed. The system switches to the normal operational mode. During this mode, the global controller takes full control and supervises all the transactions. When it receives a new packet, it directs it to its destination where the data is going to be stored. Subsequently, when one of the terminals requests a data, it reads in the respective buffer and outputs the required data. Several multiplexers have been placed at the entry of the memory blocks to select the address, data and control signals from the BIST or the global controller depending on the mode of operation. All these multiplexers are controlled by the "Test Enable" signal. Moreover, a boundary scan (JTAG) has been inserted to observe different test points.

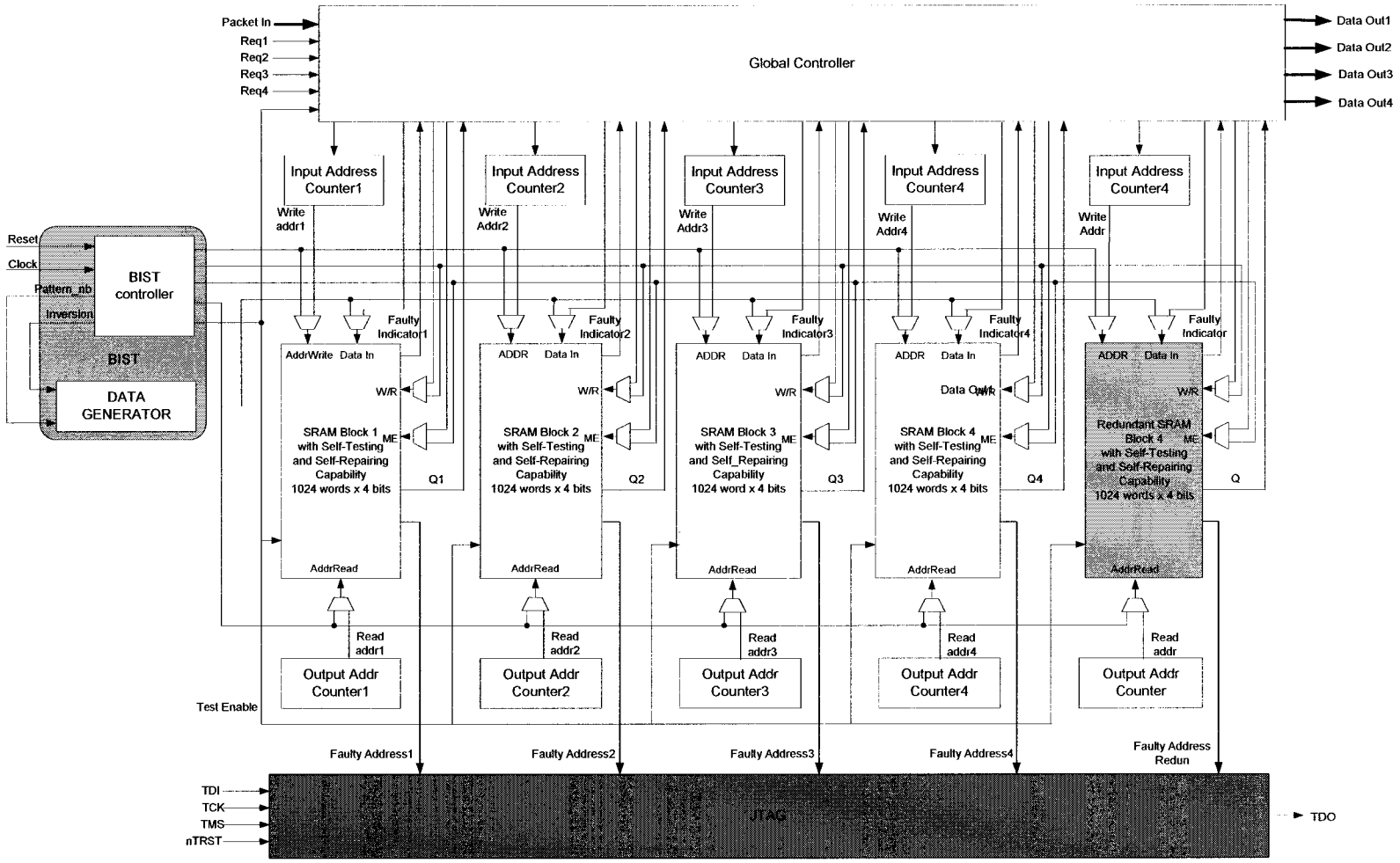


Figure 3.1 Global architecture of the implemented chip

3.2.1 Global Controller

The global controller has a double function in the proposed design. Other than implementing the FIFO functionality, such as embedded address registers, multiplexers, empty and full flags. It contains also the top level addressing block. Recall that to implement a RAM-type FIFO, a dual-port SRAM has been used to contain the data. The read and write addresses are each specified with a 10-bit counter accessing the memory via separate ports. A FF (Full Flag) and an EF (Empty Flag) are also needed to indicate the full and empty status of the FIFO. Figure 3.2 shows a functional model of this global controller. The latter controls the five memory blocks, including the redundant one. However, only one SRAM block is illustrated on this figure for simplicity.

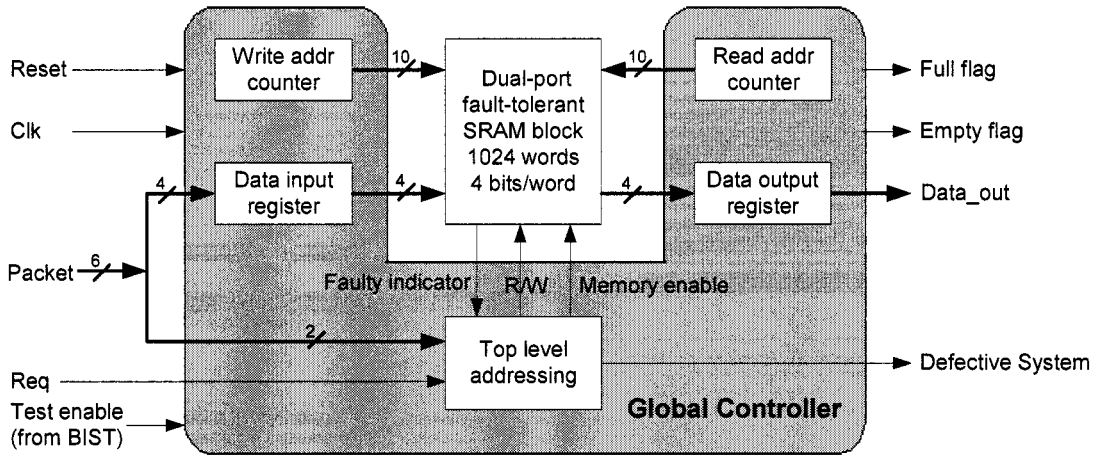


Figure 3.2 Functional model of the global controller.

When a memory block is requested for a read or a write, the top level addressing block checks its faulty indicator. If it is active, meaning that this memory block contains more

faults than the available spare rows, the top level addressing block redirects this request to the redundant block. In case there are more than one faulty blocks, the system cannot be repaired and the "Defective System" signal becomes active.

3.2.2 Boundary Scan (JTAG)

The limited pin access of the chip causes some test and diagnostic problems. In order to solve these problems, a JTAG Boundary Scan, formally known as IEEE Standard 1149.1, is introduced. It is incorporated at the Integrated Circuit (IC) level to improve its controllability and observability. The basic architecture of the boundary Scan is illustrated in Figure 3.3.

First, five package pins are dedicated to the Boundary Scan. These pins form the Test Access Port (TAP) and may not be shared with any other function. They are used with a simple protocol to communicate with on-chip Boundary Scan logic. This protocol is driven by three of the pins: Test Clock (TCK), Test Mode Select (TMS) and Test Reset (nTRST). The remaining two pins are for serially shifting data into and out of the IC, labeled Test Data In (TDI) and Test Data Out (TDO).

Second, on the IC die itself, a simple finite state machine is added called the TAP Controller. It recognizes the communication protocol and generates internal control signals used by the remainder of the Boundary Scan logic.

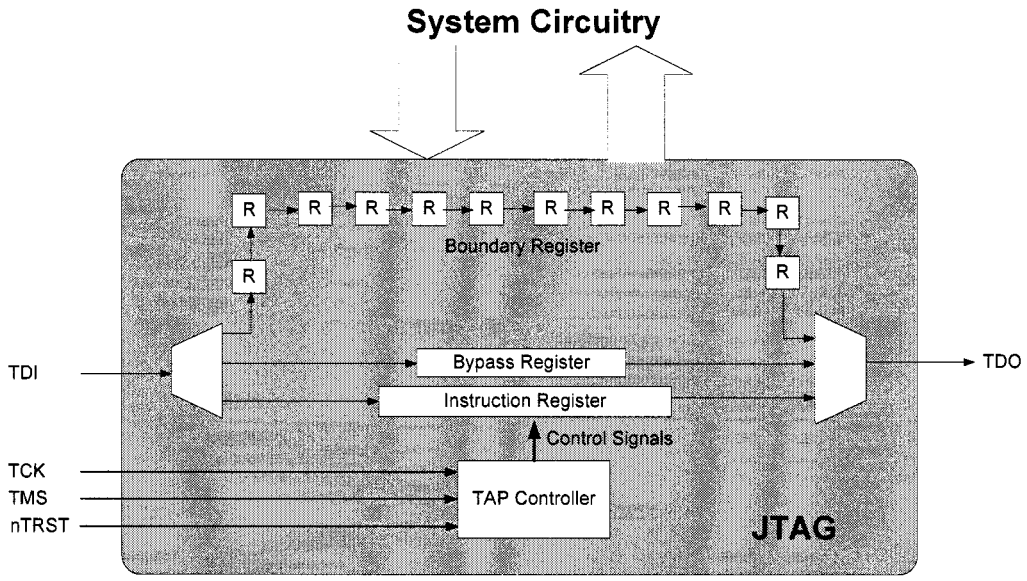


Figure 3.3 Basic architecture of the JTAG Boundary Scan.

Third, an Instruction Register can be placed between TDI and TDO for loading (and unloading) serially shifted instruction data. It is used to set the mode of operation for data registers. Then, a Bypass register consisting of only one scan cell can be selected to shorten the shift path to a single cell allowing serial data to be rapidly transferred from TDI to TDO without affecting the operation of the IC.

Last, the Boundary Register consists of a chain of boundary-scan cells. It is used to control and observe the core logic signals. Captured data from the memory blocks is serially shifted out and externally compared to the expected results. Forced test data is serially shifted into the boundary-scan cells.

3.3 THE SELF-TESTING FUNCTION

For small embedded memory, testing using serial access scanning could be acceptable. However, for a large capacity memory, this method becomes time consuming and requires storing a large amount of test data in external test equipment.

The problem of time could be solved by allowing parallel access between the IC pins and the embedded memory. But, this will cause an addition of extra pins and multiplexers and more complex routing without even solving the problem of storing lots of data in external test equipment.

A better approach to solve both problems is to provide the embedded memory with a Built-In-Self-Test (BIST) [42]. In this case, the test data are generated on chip and the test can be run at system speed. The silicon overhead required to implement the self-test logic represents the main disadvantage of this methodology.

The block diagram in Figure 3.4 represents the procedure to obtain an efficient test algorithm implemented as a BIST. First, based on the actual defects that can occur in the RAM during silicon processing, we developed a realistic fault model. At the layout level, defects are considered as logical disturbances caused by, among other factors, dust particles on the chip or masks, scratches and gate oxide pinholes. These defects are categorized according to their electrical behavior, creating a fault model at the SRAM cell level. Then, based on this model, a test algorithm that gives a fault coverage

of 100%, while keeping the area overhead and the performance degradation due to the self-test hardware at low level, has been presented and implemented.

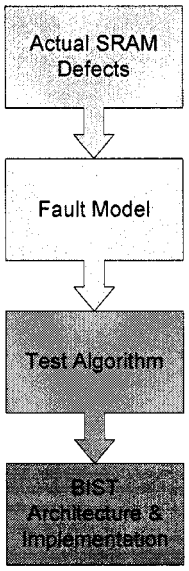


Figure 3.4 BIST implementation procedure

3.3.1 SRAM fault model

Many self-test methods and implementation for RAMs have been reported. They all use a theoretical RAM fault model. Some authors adopt a complicated fault model, resulting in irregular and complex test algorithms, which consequently leads to large silicon overhead for the self-test hardware. Others adopt a simple fault model, such as the single stuck-at fault model. However, these models do not cover many realistic RAM failure modes. We opt for the use of a more realistic model because it is based on actual defects that can occur in RAMs.

The functional model of an SRAM chip is composed of many blocks as illustrated in Figure 3.5. The actual SRAM block implemented in the proposed design will be presented in section 4.5.3. Although each block is dedicated for a special function and can become defective, faults in some blocks show the same behavior and these blocks can be grouped together as follows:

1. The memory array.

2. The address decoder: includes the address latch, the row decoder and the column decoder.

3. The R/W logic: consists of the sense amplifiers, the data registers and the write driver.

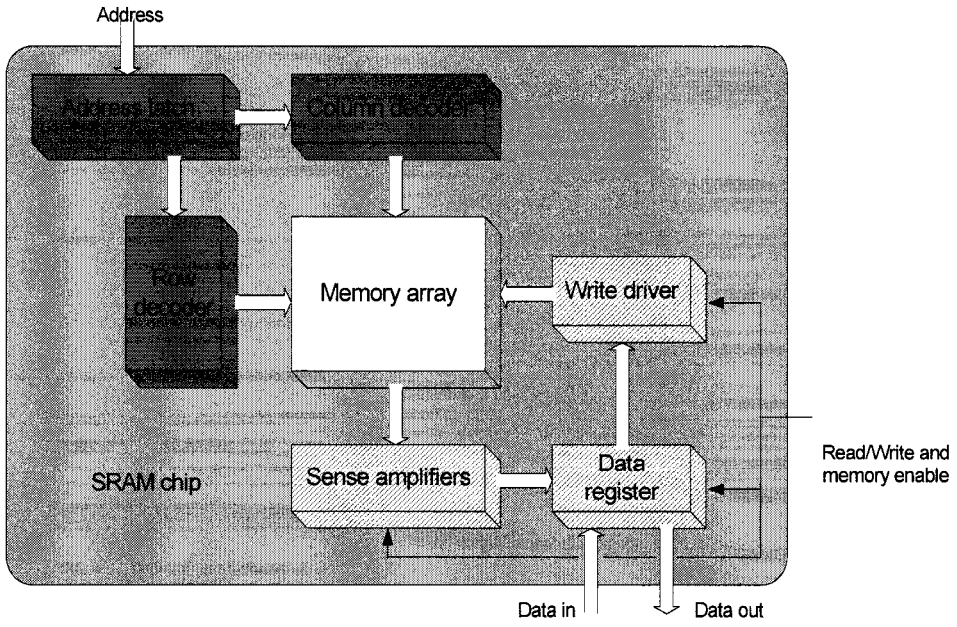


Figure 3.5 Functional model of an SRAM chip

Several faults can occur in the address decoder and they usually lead to one of the fault syndromes listed below:

- For some address, no cell is accessed.
- Some cell is not accessible.
- For some address, multiple cells are accessed simultaneously.
- Some cells are accessed with multiple addresses.

However, defects in the address decoder and the R/W logic can be mapped onto functionally equivalent faults in the memory array and therefore all the faults can be considered to be in the memory array.

The most important classes of faults occurring in SRAM array are listed below with a short explanation [13].

1. *Stuck-at fault*: A memory cell is said to be stuck-at-1 (stuck-at-0) if its content remains fixed at logic 1 (0), independently of what is written into it.

2. *Stuck-open fault*: A memory cell is said to be stuck-open if it is not possible to access the cell by any action on the cell.

3. *Transition fault*: A memory with a transition fault fails to undergo at least one of the transitions $0 \rightarrow 1$ or $1 \rightarrow 0$.

4. *Coupling fault:* A pair of cells is said to be coupled if a transition in one of them changes the content of the other cell from 0 to 1 or 1 to 0.

5. *Multiple access fault:* A memory cell is said to have multiple access fault if a write action to this cell forces a write action to another cell with the same value.

6. *Data retention fault:* There is a data retention fault in a cell if the cell fails to retain its logical value after some time period.

7. *NPSF:* A Neighborhood Pattern Sensitive Fault is a multi-cell coupling fault. It occurs when the content of a memory cell, or the ability to change the cell content, is influenced by a certain pattern of other cells in its physical neighborhood.

3.3.2 Test algorithm

The 13N test algorithm was adopted in the proposed fault-tolerant SRAM design. With this algorithm, each address is accessed 13 times, and N is the number of distinct addresses. This algorithm detects all faults of the fault model described above, including faults affecting the decoder, all stuck-at faults in the read/write logic and all single and multiple stuck-at, stuck-open, transition and non-inverting coupling faults [51].

Table 3.1 The 13N marching test algorithm for SRAMs plus data retention test.

Address	Initialization	March Element 1	March Element 2	March Element 3	March Element 4	Wait	March Element 5	Wait	March Element 6
0	Wr(0)	Rd(0),Wr(1),Rd(1)	Rd(1),Wr(0),Rd(0)	Rd(0),Wr(1),Rd(1)	Rd(1),Wr(0),Rd(0)		Rd(0),Wr(1)		Rd(1)
1	Wr(0)	Rd(0),Wr(1),Rd(1)	Rd(1),Wr(0),Rd(0)	↑	↑		Rd(0),Wr(1)		Rd(1)
2	Wr(0)	Rd(0),Wr(1),Rd(1)	Rd(1),Wr(0),Rd(0)				Rd(0),Wr(1)		Rd(1)
3	Wr(0)	Rd(0),Wr(1),Rd(1)	Rd(1),Wr(0),Rd(0)			Disable	Rd(0),Wr(1)	Disable	Rd(1)
	↓	↓	↓	Rd(0),Wr(1),Rd(1)	Rd(1),Wr(0),Rd(0)	SRAM	↓	SRAM	↓
				Rd(0),Wr(1),Rd(1)	Rd(1),Wr(0),Rd(0)				
				Rd(0),Wr(1),Rd(1)	Rd(1),Wr(0),Rd(0)				
N-1	Wr(0)	Rd(0),Wr(1),Rd(1)	Rd(1),Wr(0),Rd(0)	Rd(0),Wr(1),Rd(1)	Rd(1),Wr(0),Rd(0)		Rd(0),Wr(1)		Rd(1)

13N Test Algorithm **Data Retention Test**

The algorithm is illustrated in Table 3.1. In order to detect the data retention faults, a data retention test is added at the end, preceded by a waiting period where all memory cells are disabled. In this table, Wr indicates a RAM Write instruction and Rd indicates a Read instruction. The orientation of the data is given between parentheses. The RAM address at which the instruction is executed is shown in the first column of the table. The address order in which the instructions are performed within each March element is indicated by an arrow.

When testing a word-oriented SRAM, a read or write instruction involves reading or writing an entire word of data instead of single bits. In order to detect coupling faults between cells at the same address, we have to apply several data words, called data backgrounds, and their inversions to the SRAM during the test [50]. For m bits per word, the minimum number K of data backgrounds needed is given by equation (3.1):

$$K = \log_2 m + 1 \tag{3.1}$$

In our design, we used 4 bits per word (i.e. m = 4). According to the formula, will be 3 and the complete data backgrounds would be as follows:

Data background 1: 00 00,

Data background 2: 01 01,

Data Background 3: 00 11.

Therefore, the instructions in the test algorithm, in Table 3.1, can be interpreted as follows:

Wr (0): = Wr (data background),

Rd (0): = Rd (data background),

Wr (1): = Wr (inverted data background),

Rd (1): = Rd (inverted data background).

A complete test for a word oriented SRAM is as follows: First, the 13N test algorithm is run with the first data background, then with the second data background, and so on. Finally, both the 13N test algorithm and the data retention test are run with the last data background. There is no need to run the data retention test with all data backgrounds and it is run only once.

3.3.3 Memory Fault Coverage

The employed 13N march test algorithm detects all the faults in the fault model, i.e. it has a memory fault coverage of 100%. In this section, it will be shown that each fault is indeed covered by the 13N test algorithm.

- **Stuck-at fault**

Each cell in the memory array is checked for both states. State 0 is verified at the March element 1 and state 1 at the March element 2.

- **Stuck-open fault**

During the Rd (0) Wr (1) Rd (1) march element of the 13N test algorithm, the expected read value is alternating low and high. It is supposed to be constantly different from the latest read value, which allows detecting each stuck-open fault.

- **Transition fault**

Each cell in the memory array is checked for both transitions. The transition $0 \rightarrow 1$ is observed between the March elements 1 and 2 and the transition $1 \rightarrow 0$ between the March elements 2 and 3.

- **State coupling fault**

To detect state coupling faults, all four states of two random cells in the memory array have to be checked. This is the case with the 13N test algorithm and can be illustrated in the state diagram of figure 3.6.

- **Multiple access fault**

Assuming a multiple access fault from cell i to cell j, two march elements are sufficient to detect this fault. Two cases should be considered:

  - For j>i: (Rd(0) Wr(1) Rd(1)) and (Rd(1) Wr(0) Rd(0)) with an incrementing address order and initial state 0 are adequate.

- For j<i: the same march elements could be applied but in a decrementing address order.

Recall that all these operations are indeed executed by the 13N march test algorithm.

		Instruction	State	
			Cell1	Cell2
Initialization	1	Wr(0) cell1		
	2	Wr(0) cell2 →	0	0
March Element 1	3	Rd(0) cell1		
	4	Wr(1) cell1 →	1	0
	5	Rd(1) cell1		
	6	Rd(0) cell2		
	7	Wr(1) cell2 →	1	1
	8	Rd(1) cell2		
March Element 2	9	Rd(1) cell1		
	10	Wr(0) cell1 →	0	1
	11	Rd(0) cell1		
	12	Rd(1) cell2		
	13	Wr(0) cell2 →	0	0
	14	Rd(0) cell2		
March Element 3	15	Rd(0) cell2		
	16	Wr(1) cell2 →	0	1
	17	Rd(1) cell2		
	18	Rd(0) cell1		
	19	Wr(1) cell1 →	1	1
	20	Rd(1) cell1		
March Element 4	21	Rd(1) cell2		
	22	Wr(0) cell2 →	1	0
	23	Rd(0) cell2		
	24	Rd(1) cell1		
	25	Wr(0) cell1 →	0	0
	26	Rd(0) cell1		

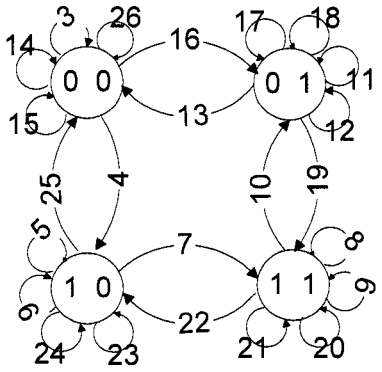


Figure 3.6 State diagram for two random cells in the memory array

3.3.4 Architecture of the BIST

Several logical blocks are needed for the implementation of the test algorithm:

- *Address counter*: generates the address sequence 0 to N-1.

- *Data generator*: Generates the true and the complement values of the data background.

- *Test data comparator*: It consists of a bitwise comparator that compares the data received from the SRAM during a Read with the expected data.

- *BIST controller*: It implements the test algorithm using a sequential circuit to generate the necessary control signals for the RAM and the other blocks of the BIST.

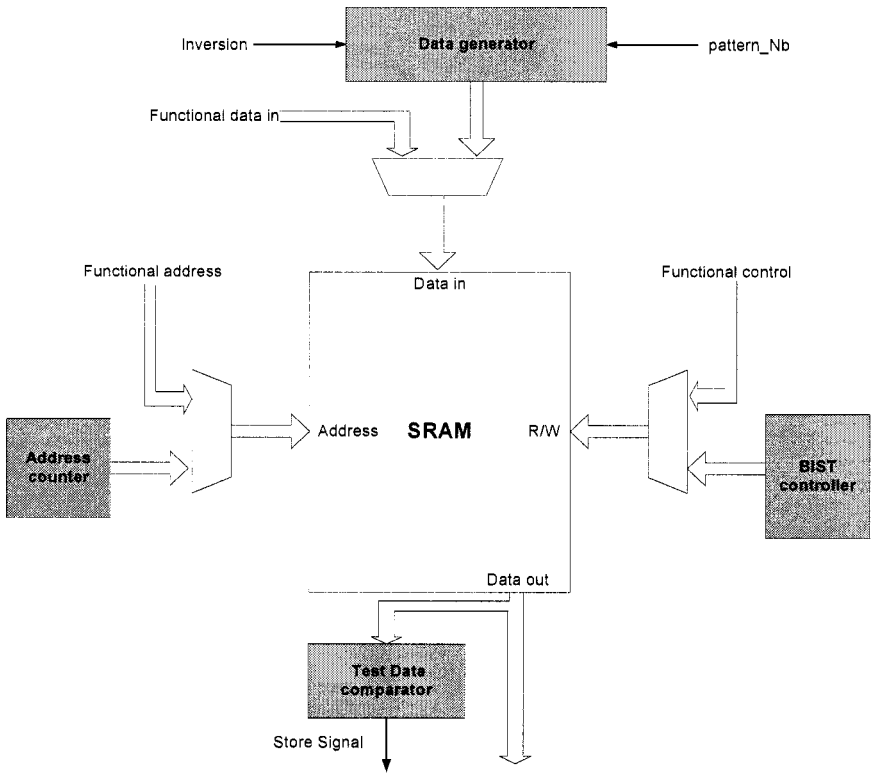


Figure 3.7 Global architecture of the self-testable SRAM.

Figure 3.7 illustrates the global architecture of the self-testable SRAM. At power on, the test runs through the address space of the memory and performs write and read operations in the order given by the test algorithm. The memory output is compared with the expected data. Any discrepancy will indicate that parts of the corresponding word are defective and need to be replaced. In this case, the BIST generates a *Store Signal* to store the faulty address in the redundancy logic.

3.3.4.1 Data Generator

The data generator generates data patterns, called the data backgrounds, to be stored in the memory according to the test algorithm. When the self-test starts, the output of the data generator is set to all 0's, representing the first data background. When the entire algorithm is completed for one word, a pulse on the *Pattern_Nb* signal, produced by the BIST controller, will activate the next data background. The *Inversion* signal, coming also from the BIST controller, switches the data generator output between the data background and its complement as required by the test algorithm.

3.3.4.2 Address counter

The address counter consists of a simple binary up/down counter that generates the address sequence from 0 to N-1, where N is the number of words in the memory array. It was incorporated inside the BIST controller. For each March element in Table 3.1, after executing the necessary operations for a given address, the controller increments or decrements the address counter as required by the test algorithm.

3.3.4.3 Test data comparator

For every read operation during the test algorithm, the test data comparator receives the data produced by the SRAM and compares it with the expected data. If there is any difference, a *Store Signal* is generated to signal that the corresponding address is faulty and it has to be stored in the self-repair logic. This verification of the correctness of the data read from the memory can also be done using a signature analysis approach, which consists of compressing the data using polynomial division [14]. However, with this method, some faults can pass undetected due to aliasing errors. Therefore, direct comparison is preferred.

3.3.4.4 BIST controller

The BIST controller initiates and stops testing and supervises the control flow of the test algorithm. During the test, it has full control. It generates all the data, addresses and control signals necessary for the execution of the test algorithm. Figure 3.8 shows a state diagram of the BIST controller implemented as a Finite State Machine (FSM).

Moreover, several multiplexers are added in order to connect the respective address, data and control signals to the memory, depending on the mode (test mode or normal operation mode).

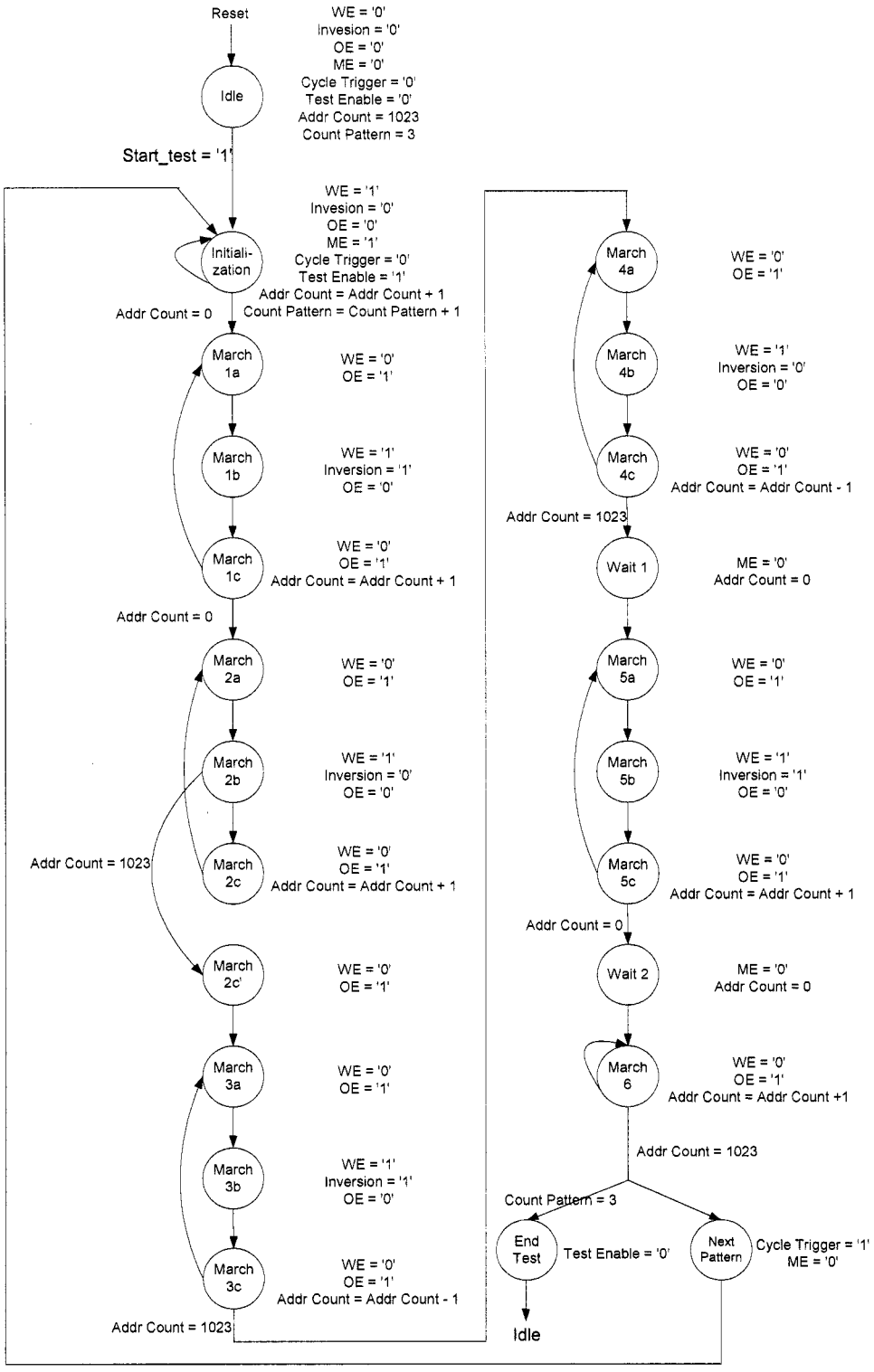


Figure 3.8 State diagram of the BIST controller

3.4 CONCLUSION

In this chapter, the fault-tolerant SRAM global architecture has been presented. It is divided into two major sections: the self-testing and the self-repairing functions. The self-repairing strategy will be treated in the subsequent chapter. A procedure to achieve an efficient BIST was adopted. It starts with developing a fault model based on the types of faults that are likely to occur in a RAM. Then, the 13N marching test algorithm was described and implemented. The BIST architecture was partitioned into four main components: the address counter, the data generator, the data comparator, and the BIST controller. Each building block was examined separately with a complete description of its operation.

Chapter 4

THE SELF-REPAIRING FUNCTION AND THE SRAM BLOCK

4.1 INTRODUCTION

In this chapter, the self-repairing component of the fault-tolerant memory design is examined. First, the basic strategy to achieve such a function is introduced. A repair algorithm is then developed. The implemented Fault-tolerant SRAM block with all its sub-blocks are depicted. Finally, a full description of the fault-repairing procedure is presented.

4.2 THE SELF-REPAIR CONFIGURATION

A two level reconfiguration strategy has been adopted. First, the SRAM is split into several blocks of the same N words x M bits size. Each block comprises two redundant additional words, i.e. each block contains N+2 words but only N of them are used. These spare words are mapped in the memory space. Assuming that the memory block contains 2^A words, where A is the number of address bits, two of these words will be dedicated for redundancy and can not be addressed by the user. This creates a hole in the memory

space which is not very convenient from a user point of view. A better approach would be to enhance the memory space by an additional redundant space. This can be achieved, for instance, by adding an extra address bit that can only be seen by the fault-tolerance mechanism. However, this will increase the system complexity. To minimize the area overhead, all the blocks are sharing the same test logic, except the test data comparator, which is incorporated in each block separately to allow a parallel fault detection.

The test is then applied for all the blocks simultaneously. If a fault is detected in one of the blocks, the defective word is replaced by a redundant one. This low-level redundancy is intended to mask small defects affecting single words in one block. If there are larger defects or several small defects distributed over more than 2 independent words within one block, it cannot be repaired and the block is declared defective. Therefore, on the higher level, a redundant block is introduced to replace a faulty one. If there are more faulty blocks than the redundant ones, the entire SRAM is considered as defective.

This strategy permits to achieve a considerable yield improvement with a rather low area overhead. Assuming a 128-Mb RAM divided into 16 blocks of 32k words by 256 bits and using the yield formulae, it was demonstrated that for more than two redundant words per block, the yield improvement is tiny compared to the additional hardware necessary for its implementation. This additional hardware can cause time delays due to the long paths connecting the memory, in addition to an increasing time required for the

reconfiguration. Therefore, it is more advantageous to use additional memory blocks on the higher level. These results will be presented in more details in section 5.4.

4.3 METHODOLOGY

Our goal is to allow the memory to function at its full capacity despite the presence of a few faulty cells. This can be done by adding redundancy to the memory and replacing these faulty cells by spare functional ones via a dynamic on-the-fly reconfiguration of the memory addressing space. This basic strategy is shown in Figure 4.1.

From an external user point of view, the memory contains a nominal addressing space of N words of M bits each. However, the memory system has a raw storage capacity of N+K words, K being the number of spare words added to the memory module for self-repairing. The address re-mapping can be achieved by using K address registers or a similar implementation, using a K-line Content Addressable Memory (CAM) for instance. Each address register A_i corresponds to a spare word S_i in the memory module. If a word W_j, $0 \leq j \leq N+K$, is found faulty by the BIST, its address will be stored at address register A_i, $0 \leq i \leq K$, and the whole word will be replaced by a spare word S_i. Note that the spare words are also tested by the BIST, as part of the memory array, and they can be repaired. Therefore, the repairable memory space includes all the N+K words of the memory.

When the user attempts to access word W_x in the memory module, its address is first looked up in the address registers and two scenarios can occur. If W_x has been previously detected as faulty during the test, its address would be stored in the address registers. The address of the replacement word S_y will be generated and will be routed to the memory module via the multiplexer. Thus, any attempt to access the faulty word W_x will be diverted to the redundant word S_y. On the other hand, if the target word W_x has never been detected as faulty, its address will not appear in the address registers and will be directly transmitted to the memory array. The controller manages all the operations of the BISR including storing the faulty addresses in the address registers making sure that the same faulty address does not get recorded twice and it is responsible of re-mapping them.

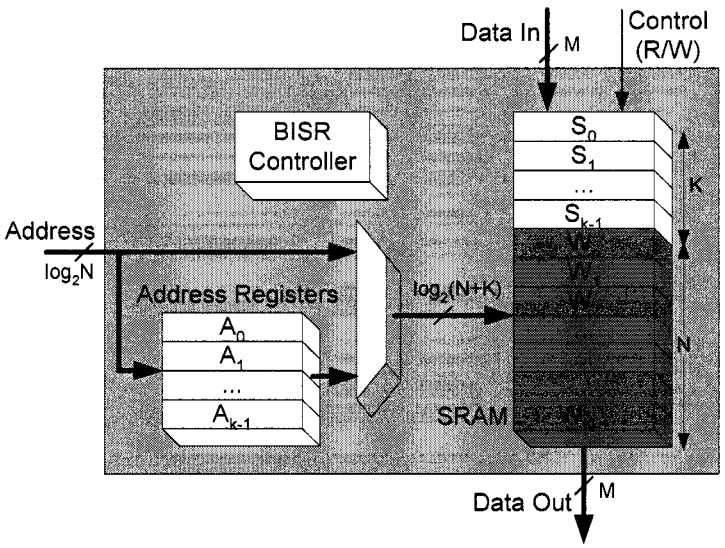


Figure 4.1 BISR basic architecture

Note that if there are more faulty words than the number of spare words, the memory module is considered as unrepairable. This module should be excluded and replaced by another module on the higher level if the architecture permits such repairs.

4.4 REPAIR ALGORITHM FLOW

The BISR controller implements the repair algorithm for replacing faulty cells. The flow chart of the spare allocation is shown in Figure 4.2.

First, the BIST and the memory are initialized. Then, the test algorithm is applied to the SRAM array, including the redundant words. If a fault is detected, the BISR controller checks that the present address has not been stored previously. If so, it allocates a free space in the registers or CAM, depending on the architecture and on previously allocated spares. Then, it records the new faulty address. In case the number of failures exceeds the number of spare words in the memory module, the controller will replace this module by a redundant one, if there is any. Otherwise, no repair is possible.

When the test terminates, the system enters the normal operation mode. In this mode, when an address is accessed, it is first searched for in the address registers (or CAM). If it corresponds to a faulty address, the replacement address of the spare word is transmitted. Otherwise, the initial address is transferred directly to the memory module.

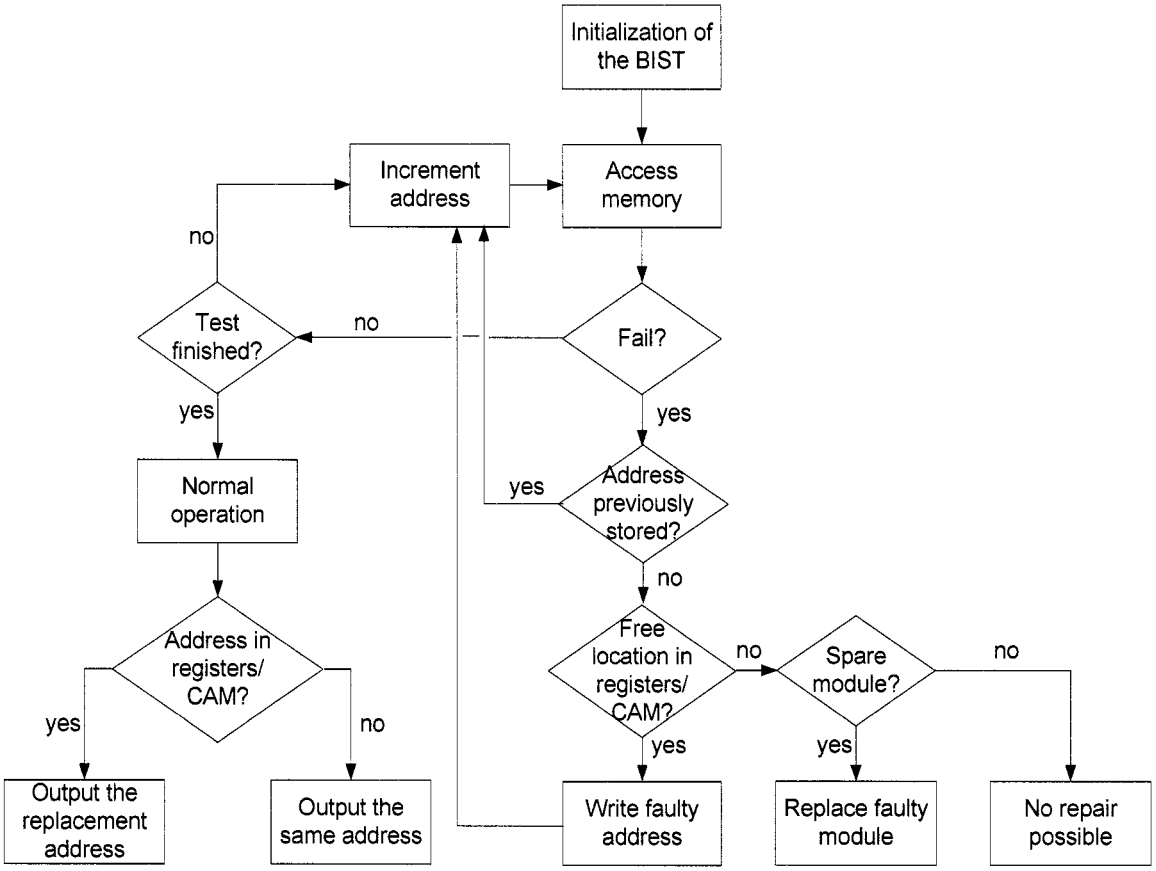


Figure 4.2 The BISR algorithm

4.5 FAULT-TOLERANT SRAM BLOCK ARCHITECTURE

The self-repairing scheme is incorporated at the lowest level of hierarchy. Each memory block is enhanced with 2 redundant words and the logic required for the reconfiguration in the case of a detected fault, as shown in Figure 4.3. This logic allows the defective word to be replaced by a redundant one.

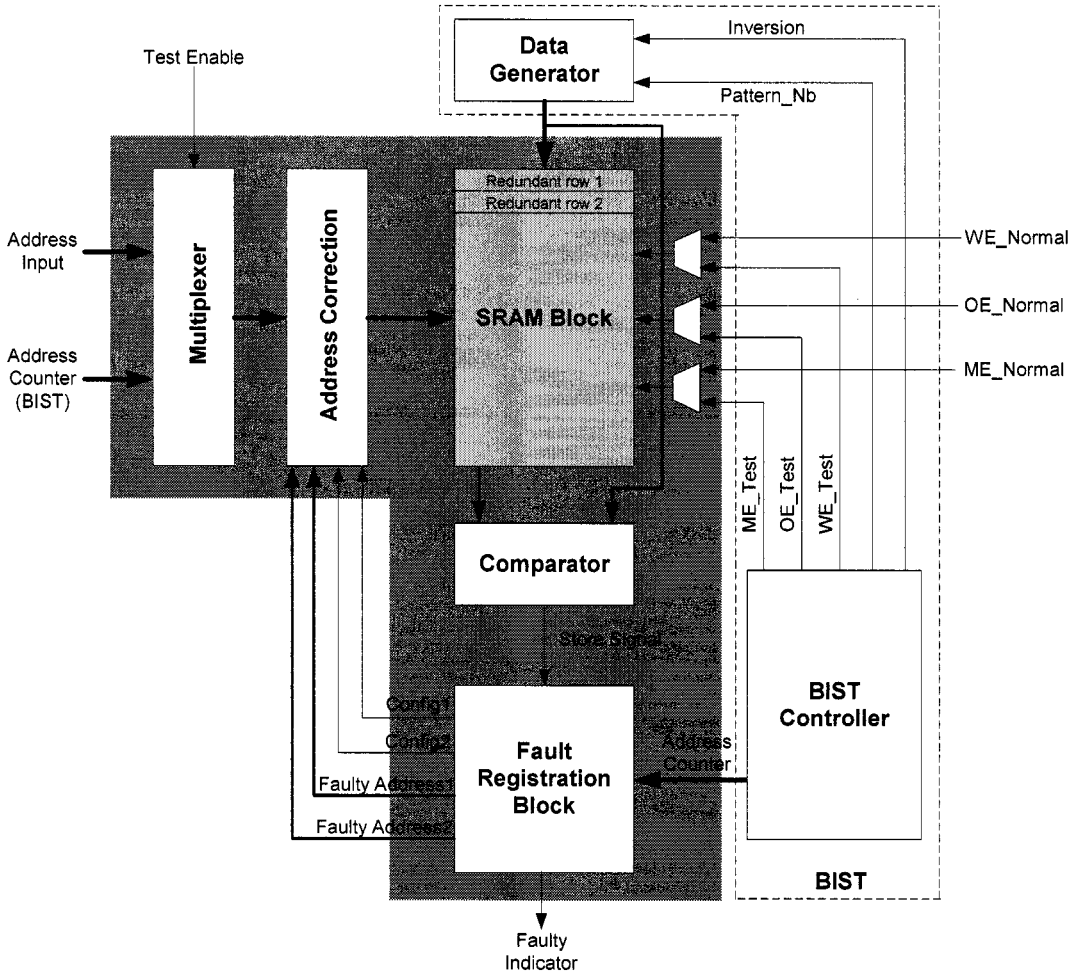


Figure 4.3 Fault-Tolerant SRAM Block.

4.5.1 Fault Registration Block

During the test, the word patterns, which are written to the memory and subsequently read out of it, are compared with the original word patterns. Any discrepancy will cause a pulse on the *Store Signal* activating the *Fault Registration Block*, which stores the memory address where the fault occurred.

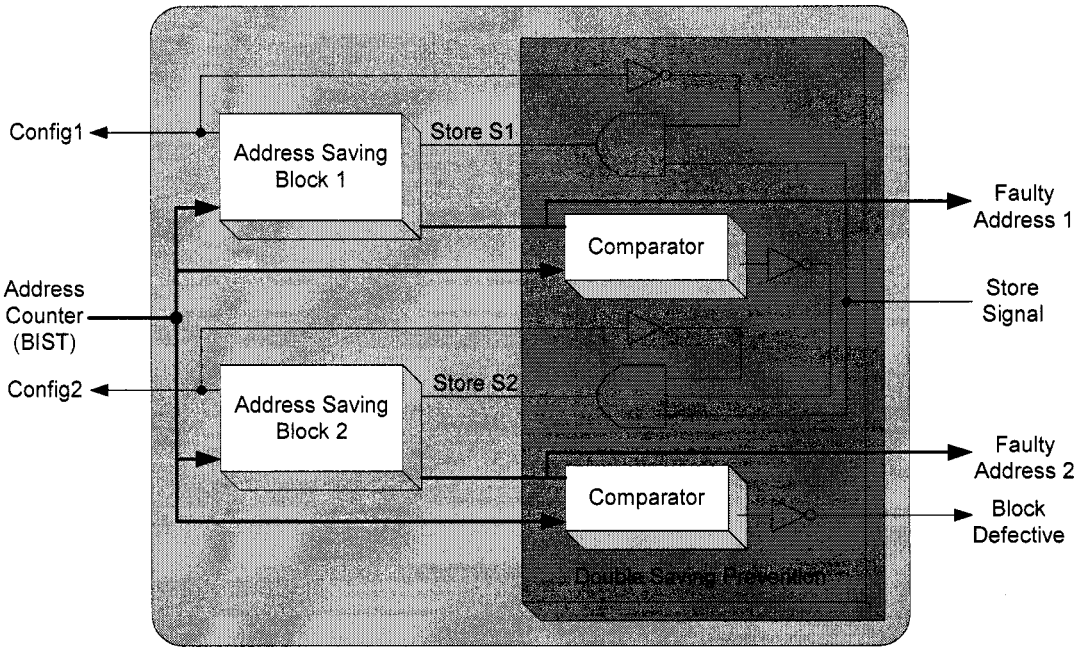


Figure 4.4 The Fault Registration Block.

The circuit shown in Figure 4.4 is capable of storing and repairing up to two faults at different memory locations. It contains two *Address Saving* blocks, illustrated in Figure 4.5. An *Address Saving* block consists of a set of address latches, where the faulty address will be stored. The signals *Config1* and *Config2* indicate if an address has been latched in the *Address Saving Block 1* or the *Address Saving Block 2* respectively.

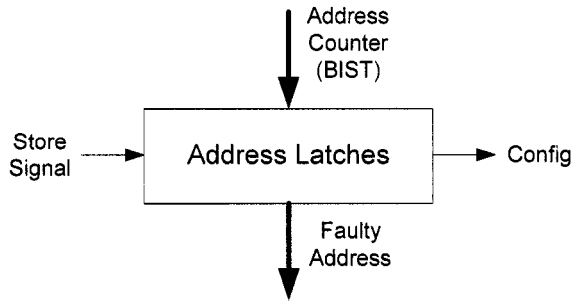


Figure 4.5 Address Saving Block

A Double Saving Prevention circuit is implemented in the Fault Registration Block to prevent the same address with multiple faults from being stored twice, that is, latched to both Address Saving blocks. The faulty address storing procedure can be illustrated in Figure 4.6.

When a fault occurs, five scenarios are expected:

1. *Address Saving Block 1* is not configured, meaning that no address has been stored in it yet. Thus, the current faulty address gets stored in this first block.

2. *Address Saving Block 1* is configured, meaning that it contains the address of a previously detected fault, and the stored address corresponds to the address counter. In this case, the current faulty address is already stored in the first *Address Saving Block* and does not need to be stored again.

3. *Address Saving Block 1* is configured, but its content is different from the address counter, and *Address Saving block 2* is not configured. Therefore, the current faulty address gets stored in the second block.

4. Both Address Saving blocks are configured and the content of the second block corresponds to the address counter. In this case, the current faulty address is already stored and no further action is needed.

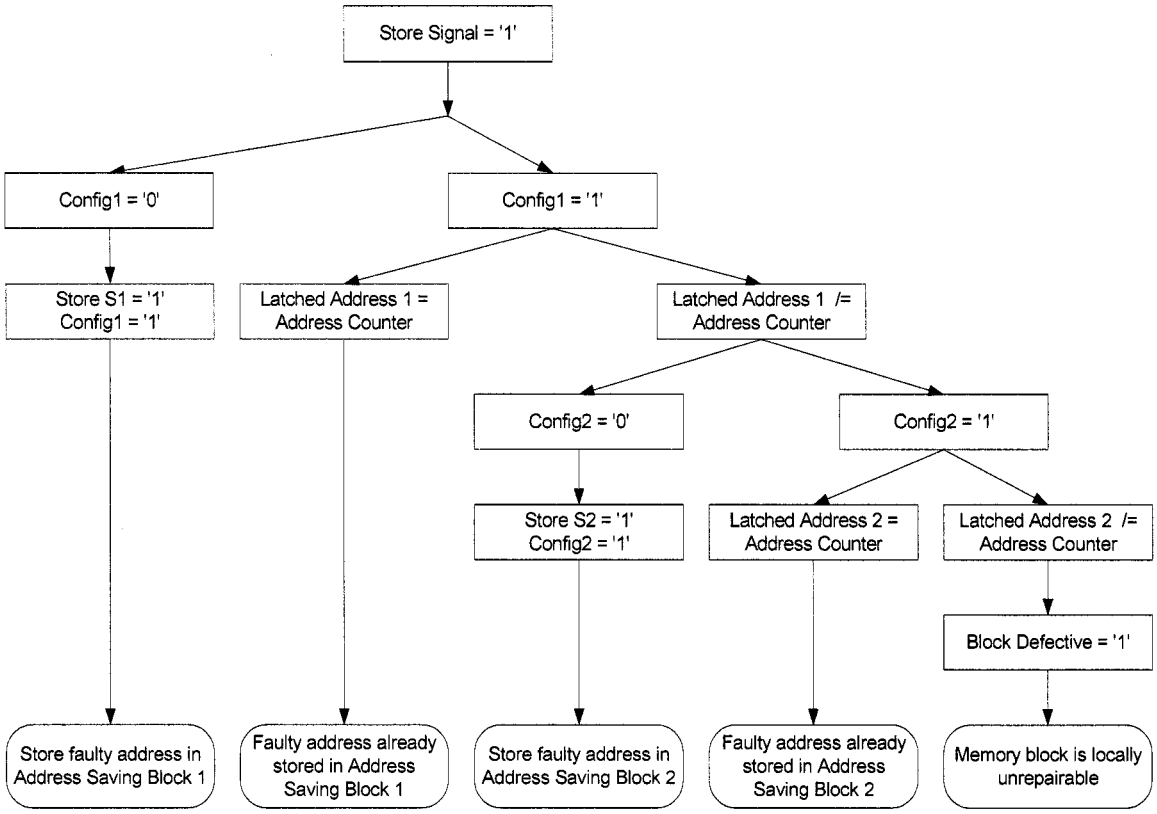


Figure 4.6 Flow for storing a faulty address

5. Both Address Saving blocks are configured, but the address counter does not correspond to any of their contents. This scenario occurs when there are more faulty words in one block than the number of spare words. In this case, the memory block cannot be repaired locally and it is considered as defective.

4.5.2 Address Correction Block

During the test mode, the address generated by the BIST passes directly through the *Address Correction* block to the address decoder inside the SRAM block. However,

during normal operation, this block is responsible of diverting an incoming address, intended for a faulty memory location, to a new address. After a fault has been diagnosed and the system switches back to normal operation mode, any incoming address will be compared with the faulty addresses stored in the self-repairing logic block called the *Fault Registration Block.* Any similarity will cause the incoming address to be redirected to a functional redundant location.

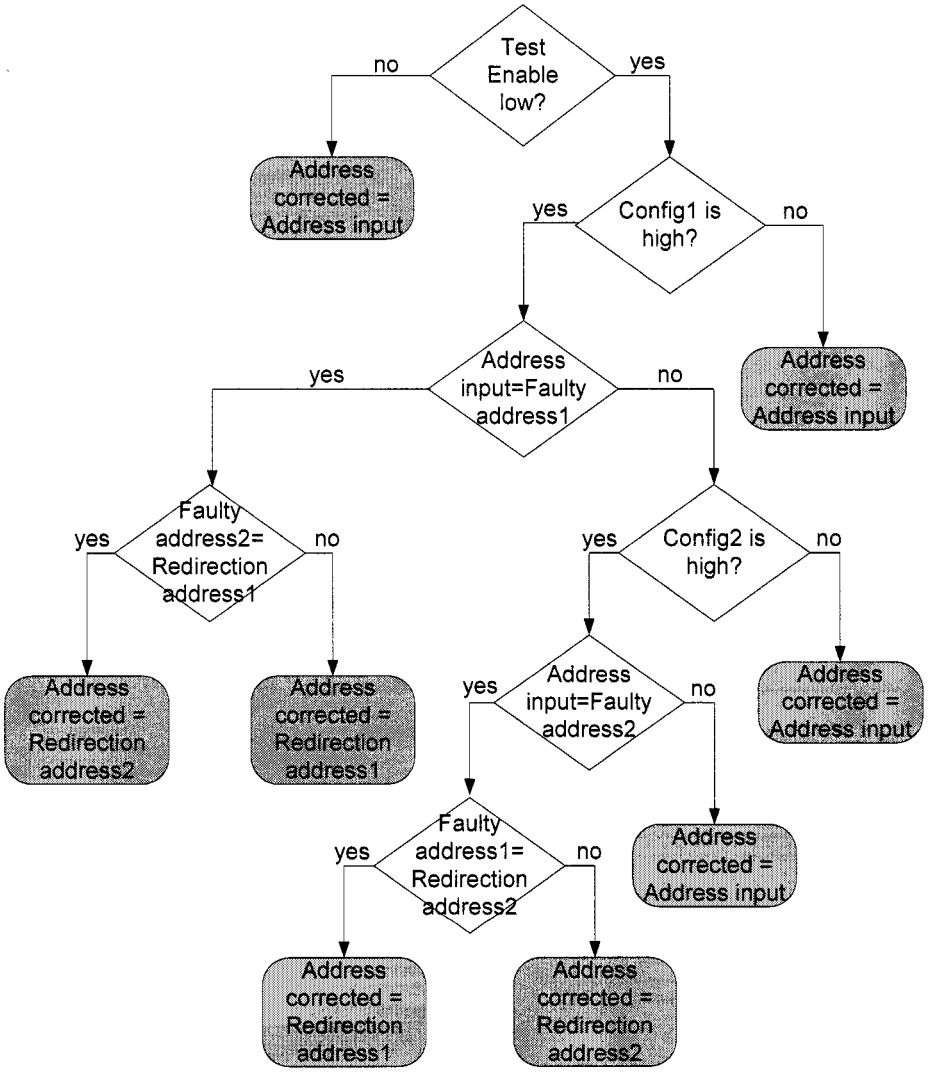


Figure 4.7 Flow of address correction.

For instance, if the incoming address corresponds to the faulty address stored in the *Address Saving Block 1*, we have to make sure that its corresponding spare location is not defective. So we compare the address of this first spare location with the address stored in the *Address Saving Block 2*. If they match, then the incoming address will be re-mapped to the address of the second spare location. Otherwise, the address of the first spare location is transmitted to the memory address decoder. This address diversion is controlled by signals generated by the *Fault Registration Block* presented in the previous section. The flow of this address correction procedure can be illustrated in Figure 4.7. The *Address Correction Block* is found on the memory's critical path and is expected to add a delay to the addressing time. This timing penalty will be analyzed in section 5.5.

4.5.3 SRAM block

Today's system-on-chip designs require many configurations of embedded memories including different sizes, aspect ratios and testing options. The RAM compilers have become of a great importance to the designers [1]. They allow them to explore various scenarios in the goal of performing an optimal floorplanning of the design. Thus, system designers can evaluate, at early stages, the architectural tradeoffs between performance, area and power consumption by simply varying different parameters of the design, such as the aspect ratio, the word depth and width. Once satisfied with the resulting configuration, the compiler creates all the model views required to drive today's most popular electronic design automation (EDA) tools. Several memory compilers exist on the market and fortunately, one of them is available in the GRM lab: The Virage Logic

Custom-Touch ASAP. This product line consists of three families of embedded memory compilers optimized for area, speed, and power to address a wide range of applications. The High-Density (HD) memories address the needs of many applications that are optimized for area; the High-Speed (HS) memories address the requirements of high-performance systems; and the Ultra-Low Power (ULP) memories address the needs of power-sensitive portable applications. These compilers offer the following relevant advantages:

- Theses easy-to-use compilers enable rapid generation of many DRC and LVS clean memory instances in minutes,
- The generated instances are fully characterized,
- The user-configurable parameters allow for the generation of a complete set of EDA views that are customized to the user's existing design flows,
- The separate Data-in/Data-out pins support a write-through mode,
- The flexible aspect ratios for area, speed, and power optimization deliver desired fit,
- The flexible number of bits per word provide for best area results,
- Zero DC Power memories can be produced for ultra-low power applications,
- The memories have either tri-state or always active outputs for maximum flexibility of design.

- Bit-write capability is available for ultimate write flexibility.

The license available in the laboratory at École Polytechnique gives access only to the following types of HD memories:

- Single-Port (1RW) Synchronous SRAM,
- Dual-Port (2RW) Synchronous SRAM,
- Diffusion, via, and metal programmable ROM.

These high-density synchronous memories target a 0.18-micron CMOS process. The quiescent current consumption is zero when the Memory Enable (ME) pin is low regardless of the activity on the clock, address or control pins [21]. The memory cores are optimized for a power supply with a voltage range of 1.6V to 2.0V. The RAM can be built with three aspect ratios for maximum area and performance optimization. Separate output (Q) and input (D) pins allow a write through cycle. The outputs are tri-state. The user has flexibility in specifying the logical size of the RAM, including both word size and number of address locations and column mux (aspect ratio).

For the purpose of this project, the Dual-Port Synchronous SRAM compiler has been selected to generate the required memories. A separate port is dedicated for each operation: one port for writing and the other for reading. The ranges for different parameters tolerated by this compiler are presented in Appendix A. The proposed design includes five memory cores of 1024 words by 4 bits each. A symbol of a single block is

shown in Figure 4.8. The input and output pins for each port are described in table 4.1. The logic truth table, timing diagrams and other characteristics of these blocks can be found in Appendix B.

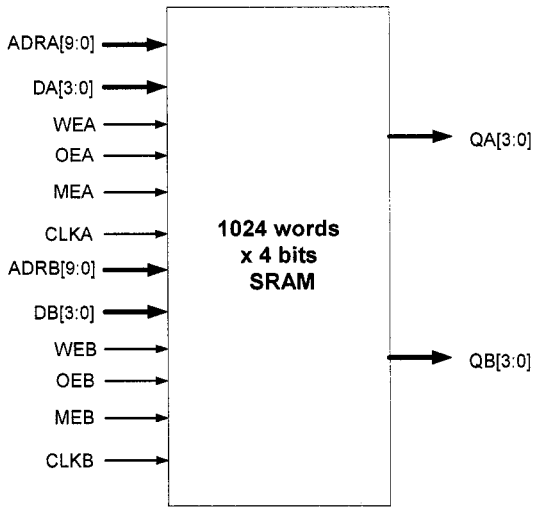


Figure 4.8 Symbol of the used memory

4.6 FAULT REPAIRING PROCEDURE

The self-repair component of the fault-tolerant SRAM design plays a specific role during each operating mode. Details of operation discussed earlier are summarized here for convenience.

4.6.1 Test mode

During the test mode, any difference between the original data and the data read out of the memory will generate a pulse on the *Store Signal*. This pulse will cause the current

address to be latched into one of the *Address Save blocks* according to the flow in Figure 4.6. Thus, two different faulty addresses (that is, memory address at which faults are present) can be recorded. A prevention circuit is used to ensure that the same faulty address will not be stored twice.

Table 4.1 Description of the input and output pins of the SRAM block

Pin	Type	Description
ADRA/ADRB	Input	Address Input This Address input port is used to address the location to be read during the Read Cycle and written during the Write Cycle.
DA/DB	Input	Data Input This is the data input bus used to write the data into the memory location specified by the address input port during the write cycle.
MEA/MEB	Input	Memory Enable Input This is the memory enable input. When memory enable is Logic High, the memory is enabled and read/write operations can be done. When memory enable is logic Low the memory is deactivated and the output goes to High-Impedance state.
WEA/WEB	Input	Write Enable Input This is the write enable input. When write enable is Logic High, the memory is in the Write Cycle Mode. When write enable is Logic Low, the memory is in the Read Cycle Mode.
OEA/OEB	Input	Output Enable Input This is the output enable control for the memory. When output enable is Logic High, Data Output indicates the contents of memory location addressed by address input. When output enable is Logic Low, Data Output is in the High-Impedance state.
CLKA/CLKB	Input	Clock Input This is the external clock for the memory.
QA/QB	Output	Data Output This is the data output bus. It is controlled by output enable as well as the memory enable. When enabled, it outputs the contents of memory location addressed by address input. When output enable or memory enable is Logic Low, Data Output is in the High-Impedance state.

The redundant words, in each memory block, are also tested by the BIST circuit. If a fault is detected in a redundant location, its address is also stored in one of the *Address Save blocks*. The two latched addresses in the *Address Save blocks* are subsequently used by the *Address Correction Block* to ensure that a faulty incoming address will be correctly redirected to a functional redundant location. For instance, if the first redundant word has been signaled as faulty, the *Address Correction Block* will make sure that any faulty incoming address will not be diverted to this first redundant word. If a third faulty address is detected, the pulse on the *Store Signal* will cause the "Defective Block" to become high, indicating that there are more faulty locations in this memory block than the number of redundant words. Therefore, this memory block is considered as defective and becomes locally unrepairable by this self-repairing circuit.

Two redundant words are introduced at the lowest level of hierarchy. Thus, at this level, this scheme can repair at most two faulty addresses in each memory block. If the number of faulty addresses is below the number of redundant words, the local self-repairing logic can repair this memory block locally and thereby restore the entire memory system to its fullest intended capacity. However, if the number of faulty address exceeds the number of redundant words, the local self-repairing logic is not capable of fixing this memory block and it must be excluded from being accessed during the normal operation mode. Any attempt to access this faulty memory block should be redirected to a functional redundant memory block at the higher level. If more than one memory block are locally unrepairable, i.e. contain more than two faulty addresses each, an output signal

"Defective System" will be generated indicating that the self-repairing logic is not able to repair all the existing faults to restore the memory system to its fullest intended capacity. However, this situation is unlikely to happen given a sufficient amount of redundancy and self-repairing logic.

4.6.2 Normal operation mode

During the normal operation mode, every incoming address is compared with the two faulty addresses stored in the *Address Save blocks*. If it matches the address in the first *Address Save Block*, the *Address Correction Block* redirects it to the first redundant location. Similarly, if it matches the address stored in the second *Address Save Block*, the *Address Correction Block* diverts it to the second redundant location. Moreover, we have to make sure that the corresponding spare location is fault free by comparing its address with the address stored in the other *Address Save Block*. For instance, if the incoming address corresponds to the address stored in the *Address Save Block 1*, then it should be normally redirected to the first redundant word. However, if the address of this latter is found in the *Address Save Block 2*, then the address of the second spare word will be transmitted instead. Thus, any incoming address is compared with at most two memory addresses where faults are present, and diverted to a new functional location if the address input matches any of these two addresses.

4.7 CONCLUSION

This chapter treated of the self-repairing part of the fault-tolerant memory design. A two-level redundancy configuration has been adopted. At the lower level, redundant words are incorporated to replace faulty ones. Spare blocks are introduced at the higher level to substitute defective ones, i.e. blocks containing more faulty words than the number of redundant words. A repair algorithm was developed based on a basic approach. A self-repairing SRAM block architecture was proposed. It is divided into three major sub-blocks in addition to several mutiplexers and a comparator: the *Address Correction Block*, the *Fault Registration Block* and the *SRAM Block* itself. These building blocks were depicted here with a detailed description of their specific operations and structures.

Chapter 5

IMPLEMENTATION AND RESULTS

5.1 INTRODUCTION

In this chapter, the prototype chip will be considered and the relevant results will be analyzed. Three issues will be discussed: area overhead, yield and timing penalty. The factors that influence each one will also be presented.

5.2 IMPLEMENTATION

A prototype memory chip of 4096 words by 4 bits including all the features discussed previously has been designed and fabricated in 0.18 μm CMOS technology. Although the chip has a relatively modest memory capacity, it incorporates all the required self-testing and self-repairing structures. The memory array is divided into 4 blocks of 1024 words by 4 bits. Each block contains two redundant words. A redundant memory block is added at the top level. The *Address Correction* and *Fault Registration* blocks are integrated into each memory block at the lowest level. However, the *Data Generator* and other control logic for the self-testing and self-repairing functions are placed at the top level and shared by all the blocks.

Note that the proposed self-testing and self-repairing logic is a very small part of the memory system. Therefore, the self-testing and self-repairing logic itself is much less vulnerable to fatal defects. However, existing solutions, such as duplication, can make this part of the functionality fault tolerant. They were not investigated further as the critical area is small in the present case.

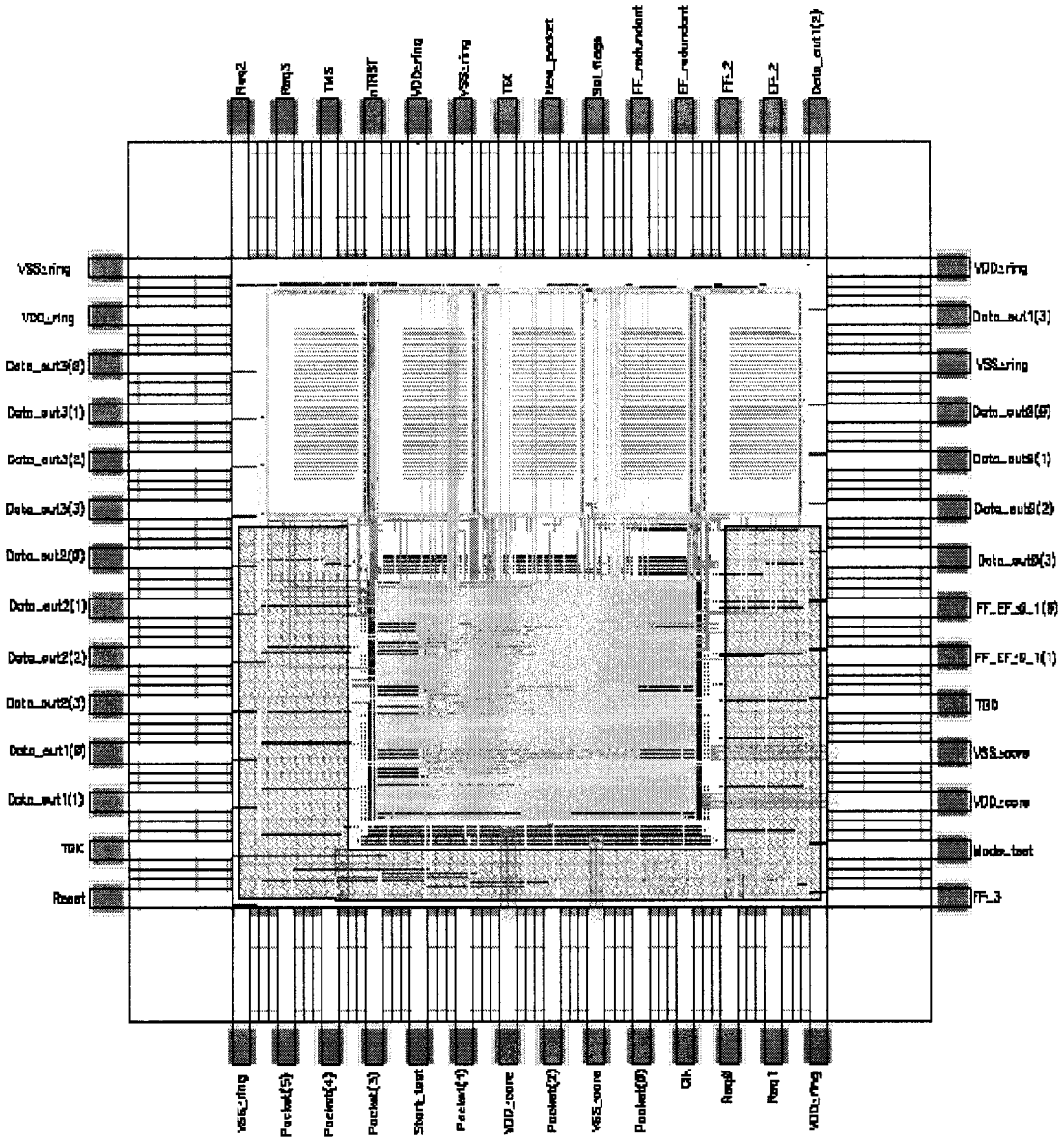


Figure 5.1 Layout view of the prototype chip.

The entire design was mapped in VHDL code, with the memory cores, created using Virage Logic memory compiler, instantiated in it. The simulations were run using Synopsys. Figure 5.1 shows a layout view of the prototype chip. The die occupies an area of 4 mm^2 and is housed in an 84-pin PGA package. This chip was fabricated by the TSMC (Taiwan Semiconductor Manufacturing Company) through the CMC. However, it could not be tested due to a design error in the output pads. Nevertheless, it was very useful because the parameters extracted from it were used as a basis for the analysis of complexity, yield and timing penalty.

5.3 AREA OVERHEAD ESTIMATIONS

Based on the actual chip layout and knowing the area of the used standard cells, investigations on the hardware overhead due to the self-test and self-repair circuitry can be carried out. The sizes of several blocks on the chip are given in Table 5.1.

Table 5.1 Sizes of several functional blocks of the fault-tolerant memory prototype chip

Block	Size (μm^2)
Data Generator and BIST Controller	15278.5
Top level blocks replacement logic	21246.8
Fault Registration block	5878.9
Address Correction block	5179.6
Muxes + Data bus	3512.7
Comparator	357.8
Memory block (1024 words x 4 bits)	106539.0

The data generator and the BIST controller at the top level are independent of the memory capacity. The sizes of the address correction block and the input and output multiplexers increase linearly with the number of address bits. The areas of the data comparator and the data bus grow linearly with the number of data bits. Moreover, the complexity of the top level replacement block increases linearly with the number of memory blocks at the lowest hierarchical level.

Thus, for the purpose of this overhead study, the self-test and self-repair logic can be divided into 4 major parts:

- X: Normalized area of the segment independent of the memory capacity,
- Y: Normalized area of the segment increasing linearly with the number of address bits per block,
- Z: Normalized area of the segment increasing linearly with the number of blocks,
- W: Normalized area of the segment increasing linearly with the number of data bits

Based on this partition, a formula for estimating the overhead of a defect-tolerant RAM has been developed and is presented in equation 5.1. This formula assumes that the RAM contains two spare rows per block at the lower level and one redundant block at the higher level.

$$\text{Overhead} = \left(\frac{A(B+1) + X + MY + BZ + NW + (B+1)(2\frac{A}{2^M})}{AB} \right) - 1 \qquad (5.1)$$

where A is the area of one memory block, M is the number of address bits per block, B is the number of blocks in the RAM, and N is the number of data bits. The numerator represents the area of the memory with repair capability and the denominator is the area of the same memory before adding such capabilities. Recall that for the fault-tolerant memory, we included one redundant memory block at the top level. Therefore, this memory contains (B+1) blocks in total. Moreover, each memory block is enhanced with two spare words. The area overhead of adding an extra word can be estimated as the area of one memory block divided by the number of words in this block ($A/2^M$).

5.3.1 Area Overhead Dependence with the RAM Block Size and the Number of RAM Blocks

To minimize the area overhead, one has to consider the block size and the number of blocks, into which the RAM is divided, as the overhead for implementing the self-test and self-repair structure depends strongly on these two factors. Figure 5.2 illustrates this dependence assuming a 4-bit wide RAM, split up into different numbers of blocks of different sizes with two redundant words per block at the local level and one spare block at the top level.

Using this redundancy configuration, the overhead decreases considerably when dividing the RAM into more blocks. The high level overhead is dominated by the area occupied by the redundant block. Therefore, the overall overhead increases significantly when the number of blocks is small. Also, it can be observed that the overhead is fairly large for small RAM sizes. Consequently, it is not advantageous to use self-testing and self-repairing strategies with small RAMs; but it is rather preferable to replace them completely.

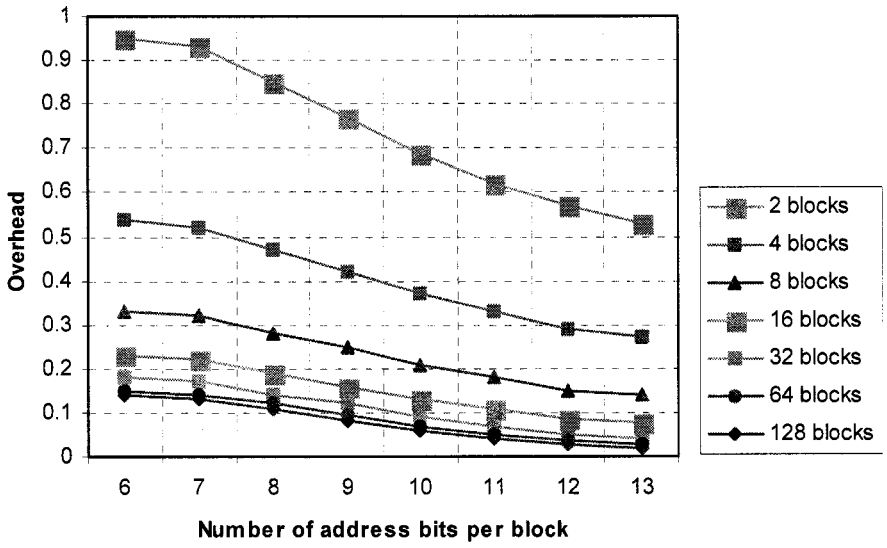


Figure 5.2 Area overhead for implementing the self-test and self-repair structure in dependence of the RAM block size for different numbers of blocks

5.3.2 Area Overhead in Dependence of the RAM Block Configuration

When implementing these self-testing and self-repairing strategies, it is not sufficient to consider the RAM size only. Its actual configuration (i.e. number of address and data

bits) has to be taken into account as well. Figure 5.3 shows the overhead associated with RAMs of different sizes, when divided into sixteen identical blocks plus one redundant block according to various possible organizations. For instance, for a 1 Mbit RAM, an organization of 2^{13} words x 128 bits results in an area overhead of 8%, compared to a 7% obtained for an organization of 2^{15} words x 32 bits.

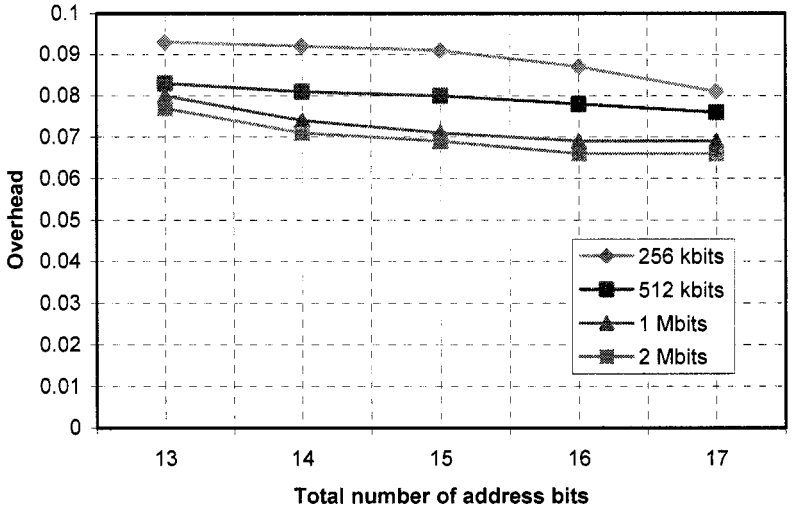


Figure 5.3 Overhead in dependence of the total number of address bits and data bits for different RAM sizes divided into sixteen blocks plus one redundant block

5.3.3 Area Overhead Projection for Ultra-Large Capacity Memory Chips

Based on the results illustrated in Table 5.1, it can be noticed that a good part of the prototype chip is occupied by the self-testing and self-repairing logic. This is due to the fact that this proof of concept IC uses small RAMs. At low memory capacity level (less than 1Mb), it is not beneficial to employ such fault-tolerance structures. It has been

applied here only for the purpose of characterizing and validating this memory architecture. Yet, the proposed design has been built in such a way that the size and the complexity of the major part of the self-testing and self-repairing structures are representative of even ultra-large capacity memory chips.

Table 5.2 Estimation of the area overhead for different memory capacities.

Memory Capacity	16 Mb	64 Mb	256Mb
Block size at the lowest level	4kb x 32	4kb x 32	4kb x 32
Number of memory blocks	128	512	2048
Number of redundant blocks at top level	1	5	20
Size of memory cells (mm^2)	114.2	456.6	1826.6
Size of redundant cells (mm^2)	0.8919	4.459	17.84
Size of address correction blocks (mm^2)	0.8018	3.647	12.85
Size of data generator and BIST controller (mm^2)	0.0153	0.0153	0.0153
Size of fault registration blocks (mm^2)	0.9100	3.039	14.59
Size of multiplexers and data bus (mm^2)	0.5438	2.179	8.717
Size of comparators (mm^2)	0.0554	0.2220	0.8879
Size of the top level replacement logic block (mm^2)	0.6799	2.720	10.88
Total area of the self-testing and self-repairing logic (mm^2)	3.0052	11.82	47.94
Area overhead excluding redundant memory cells	2.6%	2.6%	2.6%
Area overhead including redundant memory cells	3.4%	3.6%	3.6%

For instance, for large memories, the size of the lowest level memory blocks is typically in the kilobits range. Two redundant words for each of these memory blocks are usually sufficient. Therefore, the size of the fault registration block incorporated in each memory

block stays unchanged. In addition, recall that the data generator and other control logic for the self-testing function at the top level are independent of the memory capacity.

Assume a memory divided into 4 kb x 32 arrays with 2 redundant words each and 1% redundancy of these lowest level arrays at the top level. Then, based on the results obtained from the prototype chip in Table 5.1, the silicon area occupied by several functional blocks and its corresponding area overhead can be estimated for different memory capacities. These results are presented in Table 5.2.

5.4 YIELD ANALYSIS

Yield prediction is essential because too much redundancy means a waste in silicon area and too little leads to a poor yield. In order to evaluate the yield enhancement of the proposed design as well as to assess the effect of the different system parameters, several numerical calculations were performed. The yield was calculated as a function of the defect density using the equations presented in section 2.5 based on the Poisson and the negative binomial models for the defect distribution.

5.4.1 Redundant Blocks Effect

The effect of varying the number of redundant blocks is depicted in Figure 5.4. The yield improvement resulting from adding more redundant blocks at the memory top level is mostly seen at low defect densities. At high defect densities, it is observed that this

improvement in the yield becomes insignificant. The shown results assume a 128-Mb RAM divided into 16 blocks of 32k words by 256 bits with different numbers of redundant blocks at the top level.

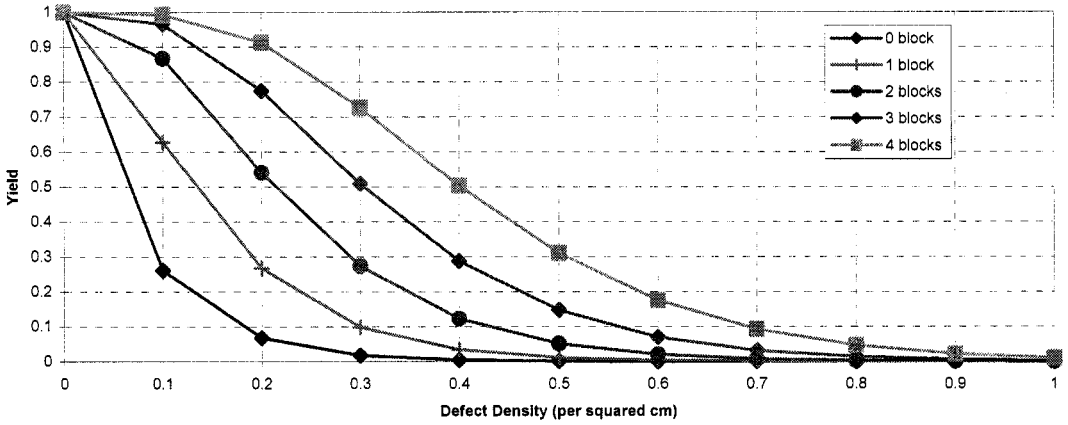


Figure 5.4 Yield as a function of the defect density for different numbers of redundant blocks.

5.4.2 Redundant Words Effect

Using redundant words only, the circuit can repair a maximum of S faulty words, S being the number of available spare words. The built-in self-repairable RAM is said to be "good" if the number of faulty words is at most equal to the number of spare words. Figure 5.5 shows the yield improvement due to BISR for a 128-Mb RAM divided into 16 blocks of 32k words by 256 bits for different numbers of redundant words. It can be observed that the total yield increases with the number of redundant words included in each block.

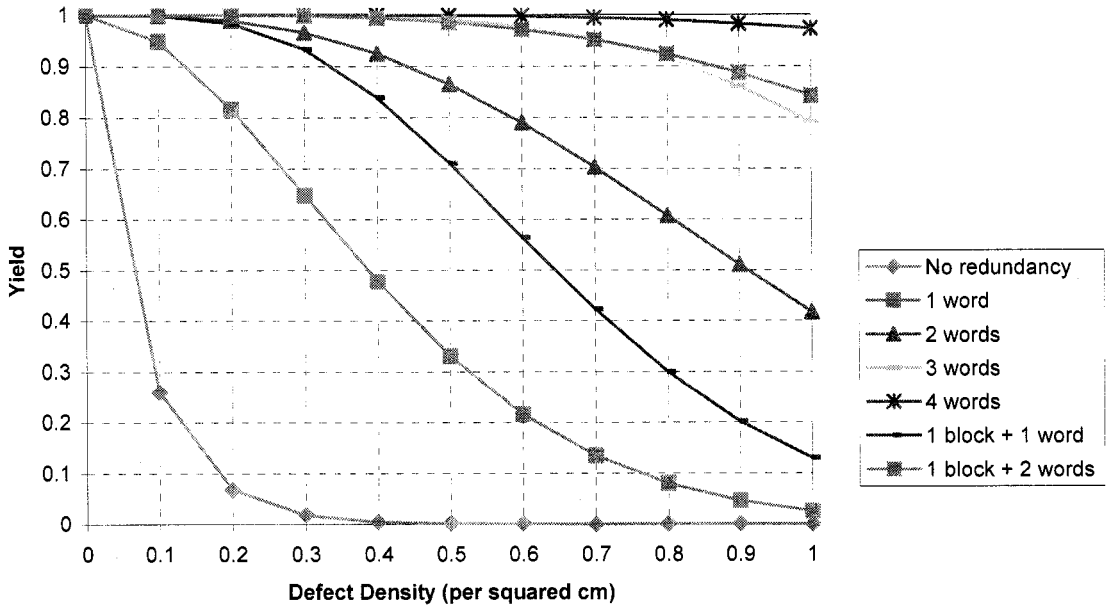


Figure 5.5 Yield as a function of the defect density for different numbers of redundant words per block and redundant blocks.

Recall that in the proposed design, row and column defects in each memory block can be detected and repaired at the low level of hierarchy or at the top level. There are no redundant columns in the local memory block. If a column defect occurs, then this memory block is considered as faulty and will be replaced by a redundant one at the top level. This column defects repair is simple but expensive. A method of using spare columns only is explained in [22]. Another redundancy approach uses spare rows and columns [7]. This method requires a complete fault mapping of the memory before determining replacement with a row or column. Therefore, the spare allocation procedure becomes quite complex.

5.4.3 Block Size Effect

Figure 5.6 illustrates the effect of dividing the memory area into several blocks. The yield enhancement due to splitting the memory into numerous smaller blocks is more significant at high defect densities. These results assume a 128-Mb RAM divided in different configurations shown in Table 5.3. For all the considered arrangements, each memory block contains the exact amount of redundancy (512 bits) distributed on two or four words depending on the number of data bits in each configuration (i.e. 2 x 256 or 4 x 128 bits) and there is no redundancy at the top level.

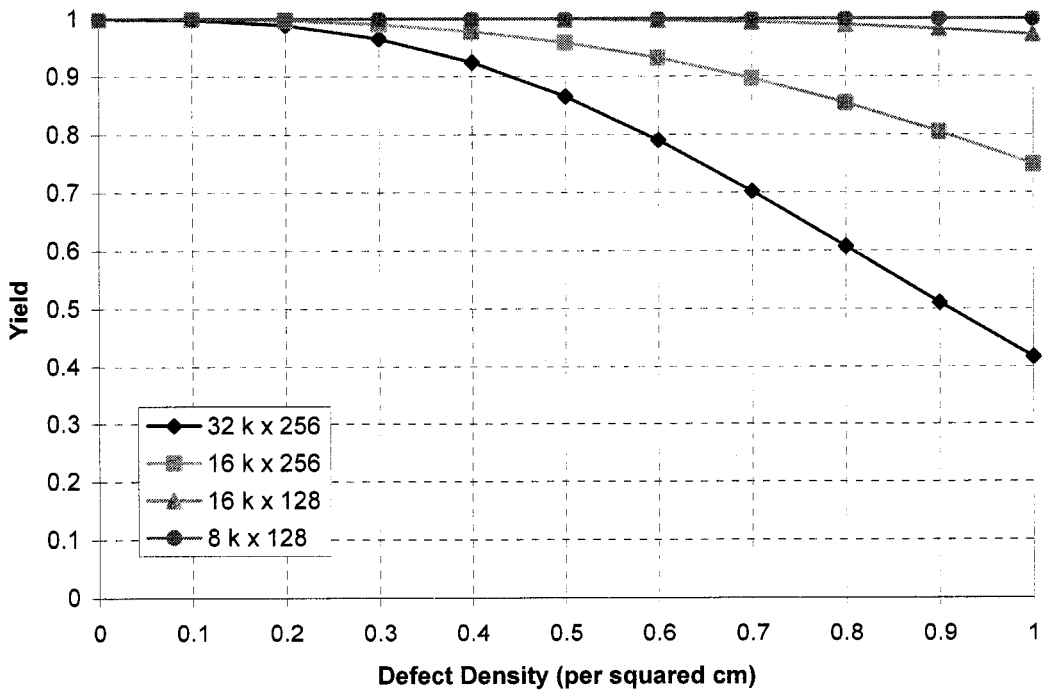


Figure 5.6 Yield as a function of the defect density for different blocks sizes.

Table 5.3 Different arrangements for a 128-Mb RAM.

Configuration	Number of blocks	Block size (bit)	Spare bits
1	16	32 k x 256	2 x 256
2	32	16 k x 256	2 x 256
3	64	16 k x 128	4 x 128
4	128	8 k x 128	4 x 128

However, having more blocks will cause an increase in the silicon area due to the added complexity to the top level addressing and the required extra wiring. This also leads to an increase in the memory access time. This fact can be illustrated in Figure 5.7 where an increase in the yield is countered by an increase in the area overhead. The challenge is to find the right compromise between both.

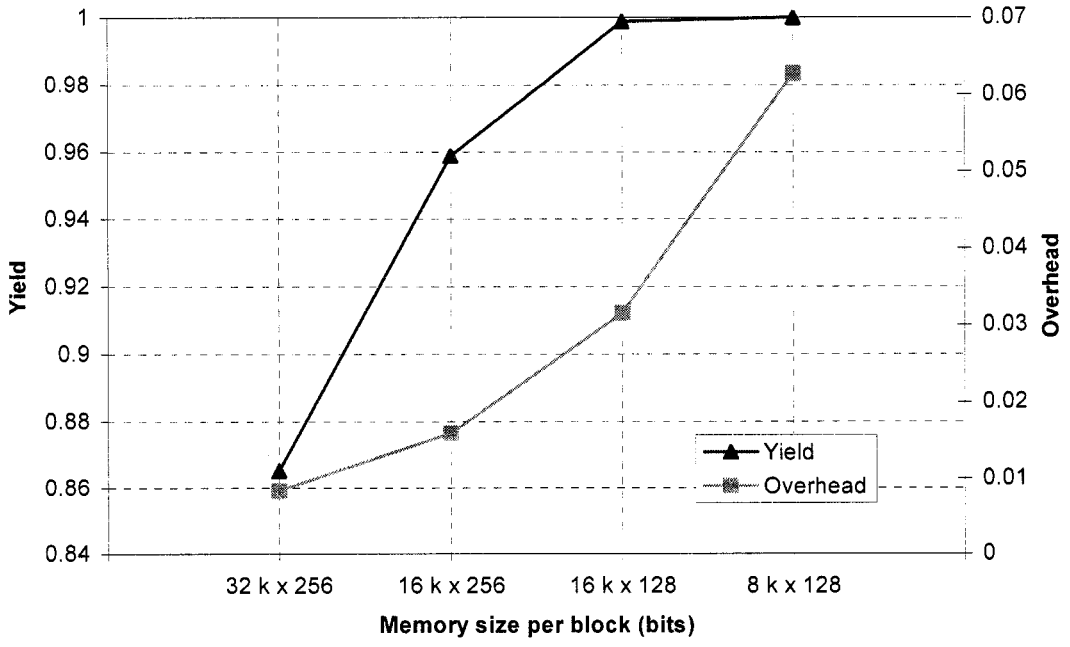


Figure 5.7 Yield improvement due to dividing the memory into smaller blocks and its corresponding area overhead.

5.5 TIMING PENALTY

As shown in Figure 5.8, some sections of the self-repairing circuit are found on the memory's critical path. This will cause a timing penalty on the memory access time. An incoming address is first selected by a multiplexer. Then, it gets compared against the faulty addresses stored in the address saving blocks located in the fault registration block. If there is a match, the address of a redundant location is generated instead of the original address for redirection. These comparison and remapping operations are performed by the address correction block. This block consists of simple random logic blocks with a maximum delay of 5 logic levels. Due to the limited number of pins available on the package, it would have been difficult to observe the signal delay through the address correction block on the prototype chip.

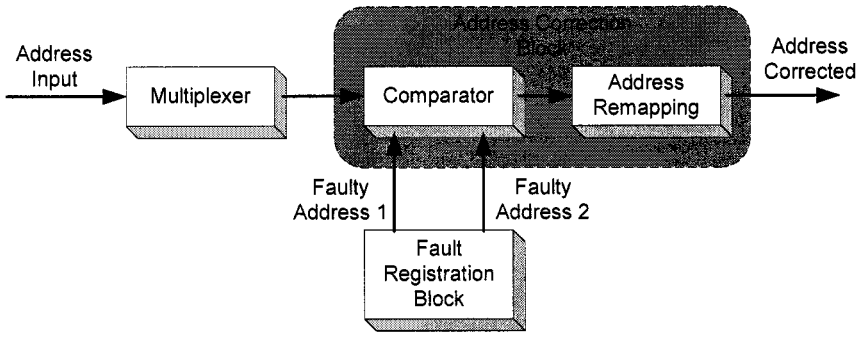


Figure 5.8 Self-repairing circuit on the memory critical path.

Figure 5.9 illustrates the time delay caused by the address correction block. Addr_inputA_normal is the incoming address and Addr_correctedA is the corrected address outputted by the address correction block. In the case where no faults were

detected at the incoming address location during the test, the simulation result indicates an extra delay of only 360 ps.

Assuming that the addresses "0X001" and "0X002" are faulty, any attempt to access one of these addresses will trigger the address correction block that then generates a replacement address to a redundant location. This redirection process adds a 900-ps delay to the addressing time. This delay is independent of the size of the memory block and the amount of redundancy inserted in it.

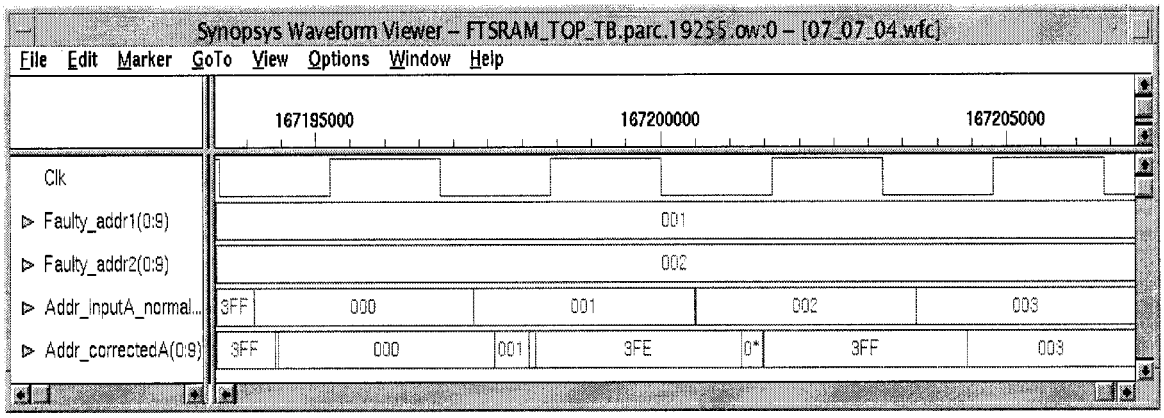


Figure 5.9 Waveform simulation showing the signal delay through the address correction block.

The top level addressing block, included in the global controller, is responsible of activating the memory block when an access to this block is requested. If this block is defective, meaning that it contains more faulty words than it can handle locally, the fault indicator should be active and the top level addressing block will redirect this request to a redundant block. This can be shown in Figure 5.10, where sram0 is defective and its fault indicator is high. When trying to access this block, the memory enable pins (MEA

and MEB) stay low indicating that it is inactive. Any read or write operation intended to this block will be transmitted to a redundant block where it will get executed. For instance, a request to read in the faulty block0 will be treated by the redundant block instead and the result will be put on the Data_out0 bus. This process is invisible to the user. The simulation result suggests maximum and minimum delays of 860 ps and 780 ps, respectively, through the top level addressing block.

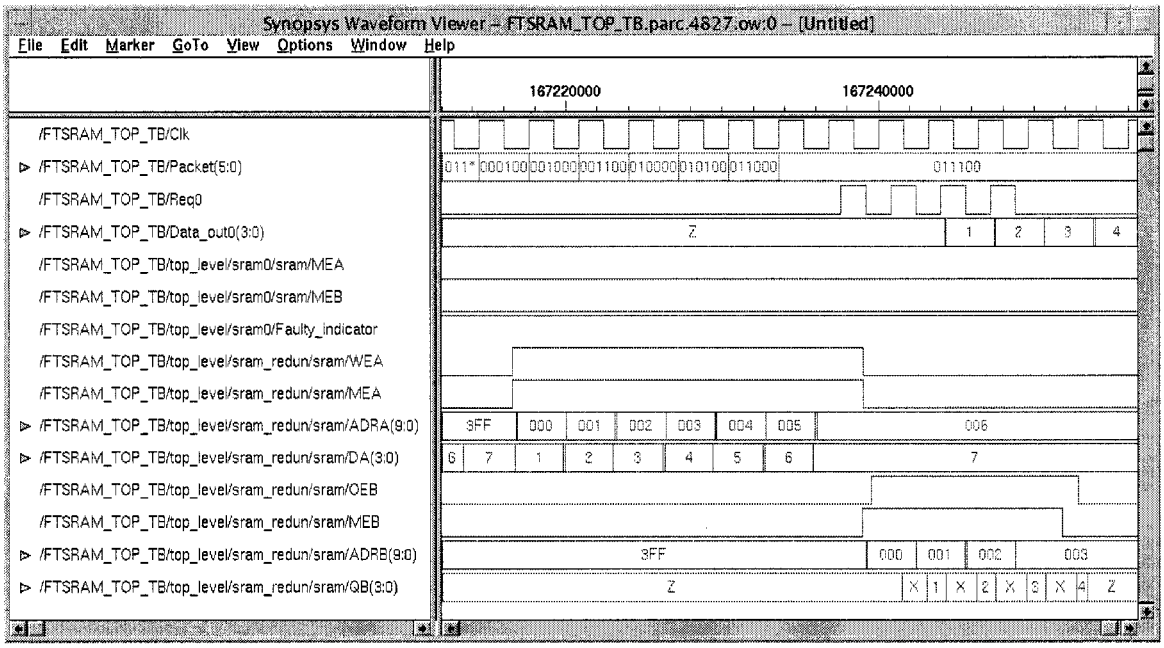


Figure 5.10 Waveform simulation showing the signal delay caused by the top level addressing block.

5.6 CONCLUSION

In this chapter, a prototype chip that was designed and implemented was characterized. Different factors that affect the area overhead, the yield and the memory access time were revealed and studied in detail. An analysis was conducted to evaluate the tradeoffs and to find the optimum compromise between these three aspects.

Chapter 6

CONCLUSION AND FUTURE WORK

In this chapter, a summary of the achieved work will be presented with its main contributions and some concluding remarks. Moreover, some improvements that can be brought to the proposed architecture will be suggested for future research.

A new built-in self-testing and self-repairing memory concept based on word replacement instead of spare rows or columns has been presented. It allows repairing RAMs generated by a memory compiler without modifying their basic architecture. This word oriented memory test and repair methodology adds two main blocks to the memory modules. Memory BIST logic implements the testing algorithm and tests the RAM at power on. Moreover, a BISR block stores the failing addresses detected during the test and replaces the faulty words. The concept of self-testing combined with self-repairing will be vital for future ultra-large capacity memory chips to reduce costs and improve the yield.

Initially, the memory array is partitioned into several identical sub-blocks. Each sub-block is provided with a dedicated logic to ensure the replacement of faulty cells and has

its own redundant words. However, since all the sub-blocks are of equal size, they share the same testing logic placed at the top level to minimize the area overhead.

This memory architecture allows a software repair in the field as long as spare words are available. Its implementation is based on RTL code and is used along with standard memories that do not have any redundancy capabilities. It enhances the memory with self-testing and self-repairing functions. Thus, the memory chip can perform tests, locate faults and repair itself without any external assistance from either test engineers or test equipment. This will increase the overall yield while reducing the production cost.

The advantages of employing such a memory design can be summarized in the following:

- Low area overhead,
- High yield,
- Soft repair,
- Repair is done on the fly,
- Scalable architecture,
- Low production cost.

The prototype chip has been evaluated according to three criteria: the area overhead, the yield and the memory access time. For each criterion, the parameters that could influence it, such as the number of RAM blocks and the redundant elements, have been presented and their effects well distinguished. It should be noted that varying one parameter might

affect several performance metrics. Therefore, care must be taken when selecting these parameters. The resulting architectural tradeoffs must be considered and evaluated to achieve the optimum solution depending on the requirements.

FUTURE WORK

In the proposed architecture, faulty words can be replaced only by redundant words included in the same memory block. However, as the number of blocks increases, the number of redundant words per block must be reduced in order to maintain the area overhead at a moderate level [52]. Consequently, the flexibility of redundancy declines.

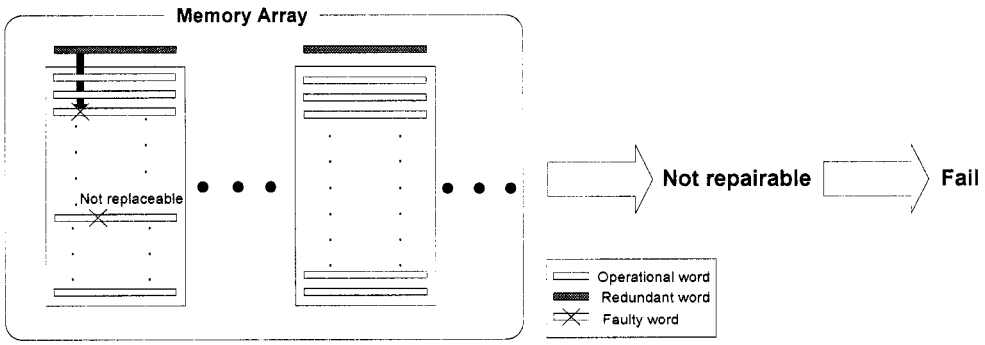


Figure 6.1 Word redundancy scheme with no flexibility.

For instance, assuming that the defects are clustered in one block, the number of faulty words will exceed the number of spare words in this block. This may cause the chip to fail, although many redundant words could be unused in other blocks. This fact can be illustrated in Figure 6.1, where the memory array is divided into several blocks and each block contains one spare word that cannot be used by other blocks.

On the other hand, multi-block division is indispensable for large memories. In this case, an inefficient redundancy usage will produce a drop in the yield. Therefore, a more flexible redundancy scheme is required for large capacity memories. An example of this scheme can be illustrated in Figure 6.2. When the number of faulty words exceeds the number of spare words in one block, the redundant words in other blocks can be used to replace the defective words. Thus, all the spare words can replace any faulty word in any block without any restriction. However, this concept may add some complexity to the design due to the fact that the address of the block has to be stored as well. Hence, a thorough study must be undertaken to evaluate this approach.

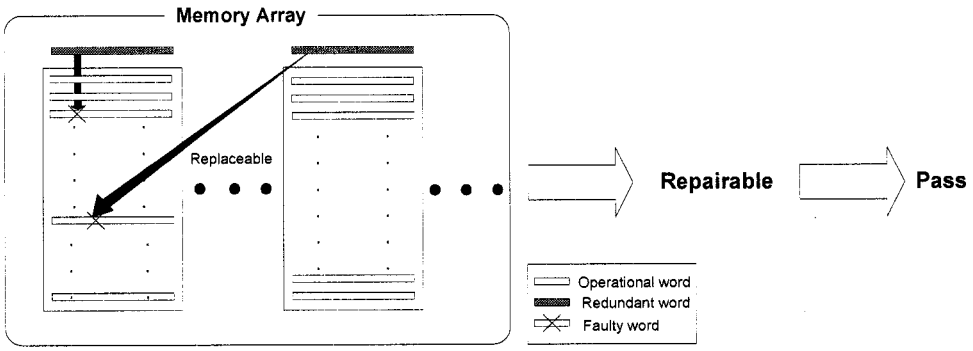


Figure 6.2 Suggested word redundancy scheme with flexibility.

Another approach is to use a Content Addressable Memory (CAM) to store the addresses of faulty words. Then, instead of using an additional register in the CAM to store the address of the redundant word, the association between the line position and the redundant word is hardwired. In other terms, each line in the CAM corresponds to a predefined location in the redundant memory space. This method allows reducing the

area and routing overhead. A sketch of this architecture is shown in Figure 6.3. Again, a study is required to prove this concept.

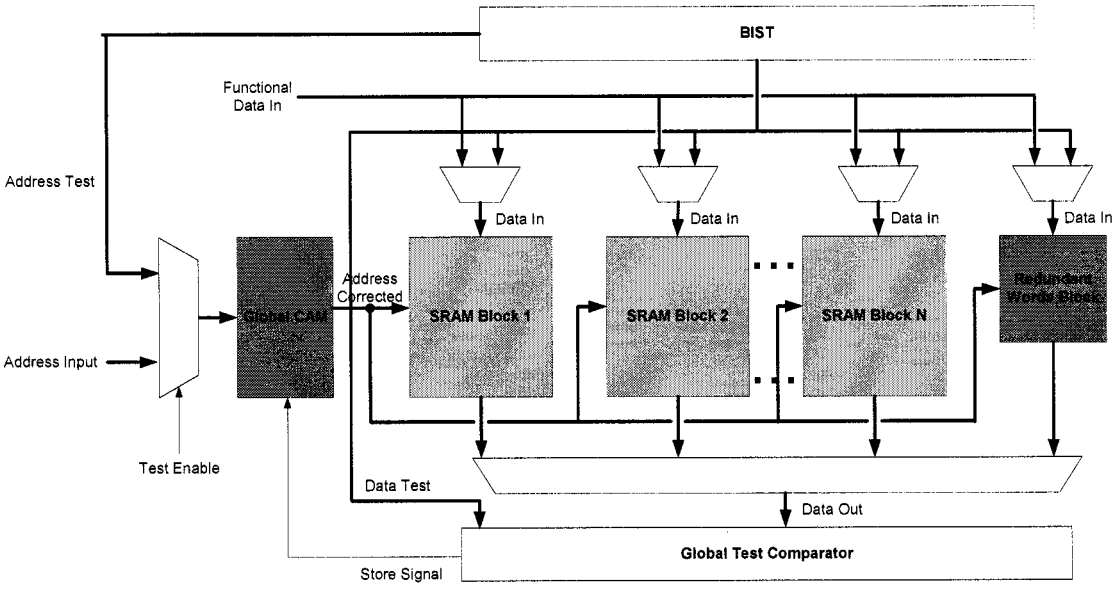


Figure 6.3 Sketch of the conceptual architecture using a CAM.

In this chapter, the main contributions of this thesis were highlighted. A brief summary of the design characteristics and advantages were presented. Finally, some suggestions to improve the architecture were proposed but need to get validated through future research.

References

1 AITKEN R., DOGRA N., GANDHI D., BECKER S., "Redundancy, repair, and test features of a 90nm embedded SRAM generator", Proc. 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, Nov. 2003, pp. 467–474.

2 BENINI L., MACCHIARULO L., MACII A., PONCINO M., "From architecture to layout: partitioned memory synthesis for embedded systems-on-chip", Proc. Design Automation Conference, June 2001, pp. 784 – 789.

3 BENINI L., MACII A., PONCINO M., "A recursive algorithm for low-power memory partitioning", Proceedings of the 2000 International Symposium on Low Power Electronics and Design, July 2000, pp. 78 – 83.

4 BENSO A., CHIUSANO S., DI NATALE G., PRINETTO P., "An on-line BIST RAM architecture with self-repair capabilities", Proc. IEEE Transactions on Reliability, Volume 51, Issue 1, March 2002, pp. 123 – 128.

5 BENSO A., CHIUSANO S., DI NATALE G., PRINETTO P., BODONI M.L., "A family of self-repair SRAM cores", Proc. IEEE On-Line Testing Workshop, July 2000, pp. 214–218.

6 BERGFELD T.J., NIGGEMEYER D., RUDNICK E.M., "Diagnostic testing of embedded memories using BIST", Proc. Design, Automation and Test in Europe Conference and Exhibition, March 2000, pp. 305 – 309.

7 BHAVSAR D.K., "An algorithm for row-column self-repair of RAMs and its implementation in the Alpha 21264", Proc. International Test Conference, Sept. 1999, pp. 311 – 318.

8 BURGESS I., "Test and Diagnosis of Embedded Memory Using BIST", EE-Evaluation Engineering, 2000.

9 CHAKRABORTY K., KULKAMI S., BHATTACHARYA M., MAZUMDER P., GUPTA A., "A physical design tool for built-in self-repairable RAMs", Proc. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Volume 9 , Issue 2 , April 2001, pp. 352 – 364.

10 CHAKRABORTY K., MAZUMDER P., "Fault-Tolerance and Reliability Techniques for High-Density Random-Access Memories", Prentice Hall PTR, 2002.

11 CHEN T., LOUDERBACK D. and SUNADA G., "Optimization of the number of levels of hierarchy in large scale hierarchy memory systems", ISCAS '92, May 1992.

12 CHOI M., PARK N., LOMBARDI F., KIM Y.B., PIURI V., "Balanced redundancy utilization in embedded memory cores for dependable systems", Proc. 17th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, Nov. 2002, pp. 419 – 427.

13 DEKKER R., BEENKER F., and THIJSSEN L., "Fault modeling and test algorithm development for static random access memories", Proc. Int. Test Conf., Sept. 1988, pp. 343-352.

14 DEKKER R., BEENKER F., THIJSSEN, L., "Realistic built-in self-test for static RAMs", Proc. IEEE Design & Test of Computers, Volume 6 , Issue 1 , Feb. 1989, pp. 26 – 34.

15 FRANKLIN M. and SALUJA K.K., "Built-In Self-Testing of Random-Access Memories", Proc. IEEE Trans. Computers, Vol. 23, No. 10, Oct. 1990, pp.45-56.

16 FRANKLIN M., SALUJA K.K., "An algorithm to test reconfigured RAMs", Proc. Seventh International Conference on VLSI Design, Jan. 1994, pp. 359 – 364.

17 FRANKLIN, M., SALUJA, K.K., "Hypergraph coloring and reconfigured RAM testing", IEEE Transactions on Computers, Vol. 43 , Issue 6 , June 1994, pp. 725 – 736.

18 FRANKLIN M., SALUJA K.K., and KINOSHITA K., "Built-In Self-Test Algorithm for Row/Column Pattern Sensitive Faults in RAMs", Proc. IEEE J. Solid-State Circuits, Vol.25, No. 2, Apr. 1990, pp. 514-524.

19 HEON-CHEOL K., DONG-SOON Y., JIN-YOUNG P., CHANG-HYUN C., "A BISR (built-in self-repair) circuit for embedded memory with multiple redundancies", Proc. 6th International Conference on VLSI and CAD, Oct. 1999, pp. 602 – 605.

20 HIROSE T., KURIYAMA H., MURAKAMI S., YUZURIHA K., MUKAI T., TSUTSUMI K., NISHIMURA Y., KOHNO Y., ANAMI K., "A 20-ns 4-Mb CMOS SRAM with hierarchical word decoding architecture", Proc. IEEE Journal of Solid-State Circuits, Vol. 25 , Issue 5 , Oct. 1990, pp. 1068 – 1074.

21 http://www.viragelogic.com

22 ILYOUNG K., ZORIAN Y., KOMORIYA G., PHAM H., HIGGINS F.P., LEWANDOWSKI J.L., "Built in self repair for embedded high density SRAM", Proc. International Test Conference, Oct. 1998, pp. 1112 – 1119.

23 KAWAGOE T., OHTANI J., NIIRO M., OOISHI T., HAMADA M., HIDAKA H., "A built-in self-repair analyzer (CRESTA) for embedded DRAMs", Proc. International Test Conference, Oct. 2000, pp. 567 – 574.

24 KIM I., ZORIAN Y., KOMORIYA G., PHAM H., HIGGINS F.P., LEWANDOWSKI J.L., "Built in self repair for embedded high density SRAM", Proc. International Test Conference, Oct. 1998, pp. 1112 – 1119.

25 KINOSHITA K. and SALUJA K.K., "Built-In Testing of Memory Using an On-Chip Compact Testing Scheme", Proc. IEEE Trans. Computers, Vol. 35, No. 10, Oct. 1986, pp. 862-870.

26 KOREN I., KOREN Z., "Yield Analysis of a Novel Scheme for Defect-Tolerant memories", Proc. IEEE Innovative Systems in Silicon Conference, 1996, pp. 269-278.

27 KOREN I., KOREN Z., "Analysis of a Hybrid Defect-Tolerance Scheme for High-Density Memory ICs", Proc. IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, Oct. 1997, pp. 166 – 174.

28 KUO S.-Y., FUCHS W.K., "Spare allocation and reconfiguration in large area VLSI", Proc. 25th ACM/IEEE Design Automation Conference, June 1988, pp. 609 – 612.

29 LU S.-K., HSU C.-H., "Built-In self-repair for divided word line memory", Proc. IEEE International Symposium on Circuits and Systems, vol. 4 , May 2001, pp. 13 – 16.

30 MAZUMDER P., YIH J.S., "A novel built-in self-repair approach to VLSI memory yield enhancement", Proc. International Test Conference, Sept. 1990, pp. 833 – 841.

31 NICOLAIDIS M., ACHOURI N., ANGHEL L., "Memory built-in self-repair for nanotechnologies", Proc. 9th IEEE On-Line Testing Symposium, July 2003, pp. 94 – 98.

32 NIGGEMEYER D., OTTERSTEDT J., REDEKER M., "A defect-tolerant DRAM employing a hierarchical redundancy scheme, built-in self-test and self-reconfiguration", Proc. International Workshop on Memory Technology, Design and Testing, Aug. 1997, pp. 33 – 40.

33 NORDHOLZ P., OTTERSTEDT J., NIGGEMEYER D, "A defect-tolerant word-oriented static RAM with built-in self-test and self-reconfiguration", Proc. Eighth Annual IEEE International Conference on Innovative Systems in Silicon, Oct. 1996, pp. 124 – 132.

34 OHTANI J., OOISHI T., KAWAGOE T., NIIRO M., MARUTA M., HIDAKA H., "A shared built-in self-repair analysis for multiple embedded memories", Proc. IEEE Conference on Custom Integrated Circuits, May 2001, pp. 187 – 190.

35 OUELLETTE M.R., ANAND D.L., JAKOBSEN P., "Shared fuse macro for multiple embedded memory devices with redundancy", Proc. IEEE Conference on Custom Integrated Circuits, May 2001, pp. 191 – 194.

36 POLIANSKIKH B., ZILIC Z., "New embedded memory architecture for enhanced yield, performance and power consumption", Proc. The 8th IEEE International Conference on Electronics, Circuits and Systems, Vol. 2 , Sept. 2001, pp. 585 – 588.

37 RONDEY E., TELLIER Y., BORRI S., "A silicon-based yield gain evaluation methodology for embedded-SRAMs with different redundancy scenarios", Proc. Eighth IEEE International On-Line Testing Workshop, July 2002, pp. 251 – 255.

38 SANGHUN P., KIJONG L., CHANGBUM I., NAMI K., KIHYUN K., YOUNGDOO C., "Designing built-in self-test circuits for embedded memories test", Proc. Second IEEE Asia Pacific Conference on ASICs, Aug. 2000, pp. 315 – 318.

39 SAWADA K., SAKURAI T., UCHINO Y., YAMADA K., "Built-in self-repair circuit for high-density ASMIC", Proc. IEEE Custom Integrated Circuits Conference, May 1989, pp. 26.1/1 - 26.1/4.

40 SCHOBER V., PAUL S., Picot O., "Memory built-in self-repair using redundant words", Proc. International Test Conference, Nov. 2001, pp. 995 – 1001.

41 STAPPER C.H., LEE H.-S., "Synergistic fault-tolerance for memory chips", Proc. IEEE Transactions on Computers, Vol. 41 , Issue 9 , Sept. 1992, pp. 1078 – 1087.

42 TANABE A., et. Al., "A 30-ns 64-Mb DRAM with built-in self-test and self-repair function", Proc. IEEE Journal of Solid-State Circuits, vol. 27, Issue 11, Nov. 1992, pp. 1525 – 1533.

43 TREUER R., AGARWAL V.K., "Built-in self-diagnosis for repairable embedded RAMs", Proc. IEEE Design & Test of Computers, vol. 10, Issue 2, June 1993, pp. 24 – 33.

44 VAN DE GOOR A.J., "Using march tests to test SRAMs", Proc. IEEE Design & Test of Computers, Vol. 10, No. 1, Mar. 1993, pp. 8-14.

45 VAN DE GOOR A.J., SCHANSTRA I., ZORIAN Y., "BIST for ring-address SRAM-type FIFOs", Proc. IEEE International Workshop on Memory Technology, Design and Testing, Aug. 1994, pp. 112 – 118.

46 VAN DE GOOR A.J. AND TLILI I.B.S., "March tests for word-oriented memories", Design, Automation and Test in Europe, 1998., Proceedings 23-26, Feb. 1998, pp. 501-508.

47 VAN DE GOOR A.J. AND VERRUIJT C.A., "Overview of deterministic Functional RAM Chip Testing", ACM Computing Surveys, Vol. 22, No. 1, Mar. 1990, pp. 5-33.

48 VAN DE GOOR A.J., ZORIAN Y., "Fault models and tests specific for FIFO functionality", Proc. IEEE International Workshop on Memory Testing, Aug. 1993, pp. 72 – 76.

49 VARDANIAN V.A., ZORIAN Y., "A March-based fault location algorithm for static random access memories", Proc. IEEE International Workshop on Memory Technology, Design and Testing, July 2002, pp. 62 – 67.

50 WANG W.-L., LEE K.-J., "A programmable data background generator for march based memory testing", Proc. IEEE Asia-Pacific Conference on ASIC, Aug. 2002, pp. 347 – 350.

51 WANG W.-L., LEE K.-J., Wang J.-F., "An on-chip march pattern generator for testing embedded memory cores", Proc. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 9 , Issue 5 , Oct. 2001, pp. 730 – 735.

52 YAMAGATA T., SATO H., FUJITA K., NISHIMURA Y., ANAMI K., "A distributed globally replaceable redundancy scheme for sub-half-micron ULSI memories and beyond", Proc. IEEE Journal of Solid-State Circuits, vol. 31, Issue 2 , Feb. 1996, pp. 195 – 201.

53 ZORIAN Y., "Yield improvement and repair trade-off for large embedded memories", Proc. Design, Automation and Test in Europe Conference and Exhibition, March 2000, pp. 69 – 70.

54 ZORIAN Y., SHOUKOURIAN S., "Embedded-memory test and repair: infrastructure IP for SoC yield", Proc. IEEE Design & Test of Computers, Volume 20 , Issue 3, May-June 2003, pp. 58 – 66.

55 ZARRINEH K., ADAMS R.D., ECKENRODE T.J., GREGOR S.P., "Self test architecture for testing complex memory structures", Proc. International Test Conference, Oct. 2000, pp. 547 – 556.

APPENDIX A

COMPILER PARAMETER RANGES

Parameter	Minimum	Maximum	Increment
Address inputs	4	14	1
Address locations (words)	16	16k	1 x CM*
Word size (number of IO-bits)	2	128	1-bit
Total memory bits	32	512k	

* CM = 4, 8, 16 : Column Mux option (aspect ratio)

APPENDIX B

MEMORY BLOCK CHARACTERISTICS

****

*** High Performance 2P(2rw) Sync High Density RAM (Rev B04p8) ***

*** Technology: TSMC 0.18u CMOS Process ***

Memory Name : hdss2_1024x4cm16

Memory Size : 1024 words x 4 bits

Column Mux Option: 16

Frequency : 300 Mhz

Memory Features:

- High Performance 2-Port (2rw) Sync High Density RAM.
- Control Signals can be configured active high or active low.
- Zero quiescent current.
- Three aspect ratios for optimization.
- Separate data-in, data-out pins to support a write through feature.

Output Ports:	PORT NAME	PIN CAPACITANCE
	QA [3:0]	0.028 pf
	QB [3:0]	0.028 pf
Input Ports:	ADRA [9:0]	0.023 pf
	DA [3:0]	0.008 pf
	WEA	0.021 pf
	OEA	0.022 pf
	MEA	0.022 pf
	CLKA	0.113 pf
	ADRB [9:0]	0.023 pf
	DB [3:0]	0.008 pf
	WEB	0.021 pf
	OEB	0.022 pf
	MEB	0.022 pf
	CLKB	0.113 pf

Memory Area: 225.680 x 472.080 = 106539.0144 square microns

Memory Internal Area: 210.23 x 449.38 = 94473.2 square microns

Bottom ring width: 3.000 microns

Logic Truth Table

(The Tables below show the Active High Control Signal Option)

For PORT A or PORT B

Function	CLKA	MEA	WEA	OEA	DA	QA
Idle	L	X	X	L/H	X	Z/Q-1
Disabled	H	L	X	L/H	X	Z
Read	H	H	L	H	X	Data-out
Write	H	H	H	L/H	Data-in	Z/Data-out
Tri-state	H	H	X	L	X	Z

A. Dual port Contention Issue (Port A address equal to port B address)

Port A	Port B	DA	DB	QA	QB	Memory State
Read	Read	DA	DB	Mem[a]	Mem[a]	No Change
Write	Read	DA	DB	DA	Unknown	Mem[a]<=DA
Read	Write	DA	DB	Unknown	DB	Mem[a]<=DB
Write	Write	DA	DB	DA	DB	Mem[a]<=Unknown

B. Dual port Contention Issue (Port A address not equal to port B address)

Port A	Port B	DA	DB	QA	QB	Memory State
Read	Read	DA	DB	Mem[a]	Mem[a]	No Change
Write	Read	DA	DB	DA	Mem[b]	Mem[a]<=DA
Read	Write	DA	DB	Mem[a]	DB	Mem[b]<=DB
Write	Write	DA	DB	DA	DB	Mem[a]<= DA Mem[b]<=DB

Operating Conditions

PVT corner	Process	Voltage(v)	Temperature(C)
Best	B	1.98	-40
Typical	T	1.8	25
Worst	W	1.62	125

Timing Characterization

Timing tables are based on 5 input slew rates and 5 output loads. Path delays are modeled with 5x5 tables, based upon 5 output loads and 5 input slew rates. Setup and Hold timing is modeled with 5x5 tables, based upon 5 clock slew rates and 5 signal slew rates. Slew rates are measured from 10% to 90% of the power supply. All timing is measured from a logic threshold at 50% of the power supply. Timing is evaluated at three timing conditions to produce three timing libraries with worst, typical and best case timing.

Look Up Table (5x5)

Index	1	2	3	4	5
Input Slope (ns,10-90%)	0.010	0.050	0.400	1.000	2.000
Output Load(pF)	0.010	0.050	0.150	0.500	2.000

Read and Write Cycle Timings

Description	Symbol	Condition	Best	Typical	Worst
CLK Low Cycle Width	Tcl	Min	0.369	0.560	0.909
CLK High Cycle Width	Tch	Min	0.346	0.526	0.853
CLK Cycle Time	Tcc	Min	1.138	1.726	2.801
CLK to Q Delay	Tcq	Max	1.126	1.709	2.773
Q hold time after CLK High/Low	Tcqx	Min	0.416	0.631	1.024
CLK High/Low to High Z Q delay	Tmqz	Max	0.302	0.459	0.744
CLK High to High/Low Q delay	Tmq	Max	0.480	|0.729	1.182
OE High/Low to High Z Q delay	Toqz	Max	0.231	|0.351	0.569
OE High to High/Low Q delay	Toq	Max	0.223	0.338	0.549
ADR setup time before CLK rises	Tac	Min	0.344	0.347	0.401
ADR hold time after CLK rises	Tcax	Min	0.000	0.000	0.000
D setup time before CLK rises	Tdc	Min	0.345	|0.352	0.570
D hold time after CLK rises	Tcdx	Min	0.013	0.012	0.011
WE setup time before CLK rises	Twc	Min	0.342	0.351	0.355
WE hold time after CLK rises	Tcwx	Min	0.000	|0.000	0.000
ME setup time before CLK rises	Tmc	Min	0.347	0.344	0.364
ME hold time after CLK rises	Tcmx	Min	0.000	0.000	0.000
Posedge CLKA recovery w.r.t Posedge CLKB	Tcsep	Min	1.138	1.726	|2.801
Posedge CLKB recovery w.r.t Posedge CLKA	Tcsep	Min	1.138	1.726	2.801

Note:

1. The data in the above table are specified with no additional load on the output pins. To obtain Tcq and Toq timing under a specified load (CL), use the following equations:

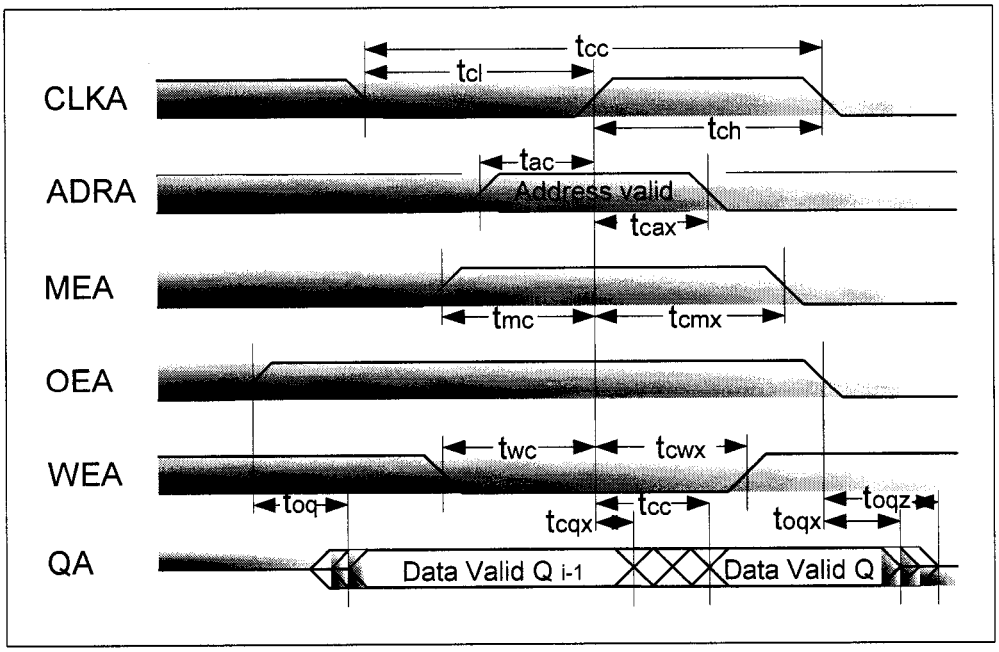
   Tcq with load = Tcq + (Ktd * CL)
   Toq with load = Toq + (Ktd * CL)

   Here, value of Ktd is as per the table below.

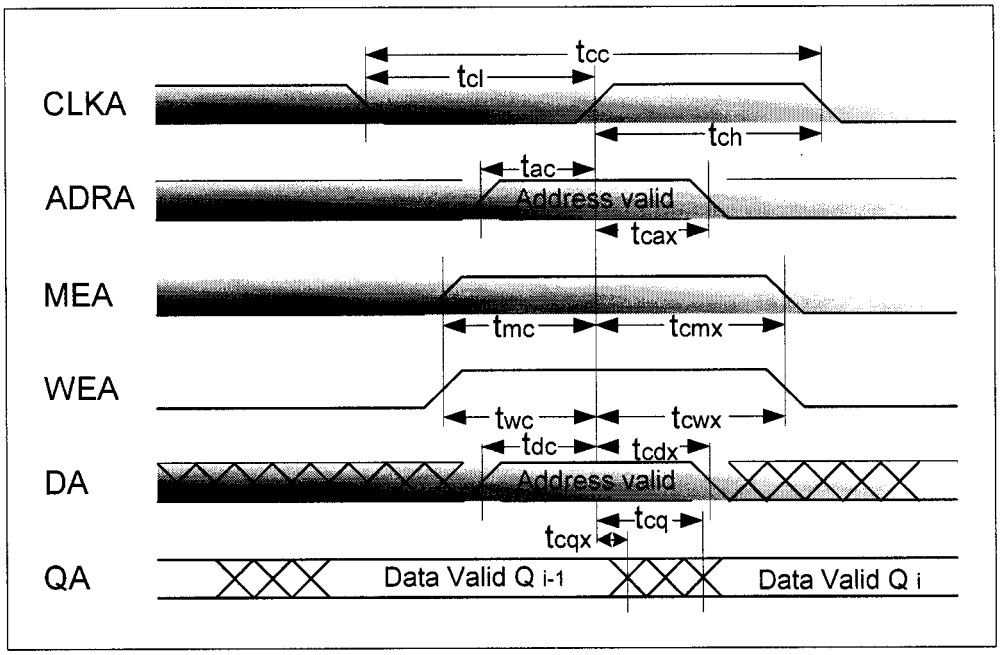
Description	Symbol	Best	Typical	Worst
Slope ns/pf in the extrinsic delay	Ktd	0.279	0.386	0.578

2. Memory Enable setup time and Address setup time in the above table have a minimum value regardless of PVT condition.

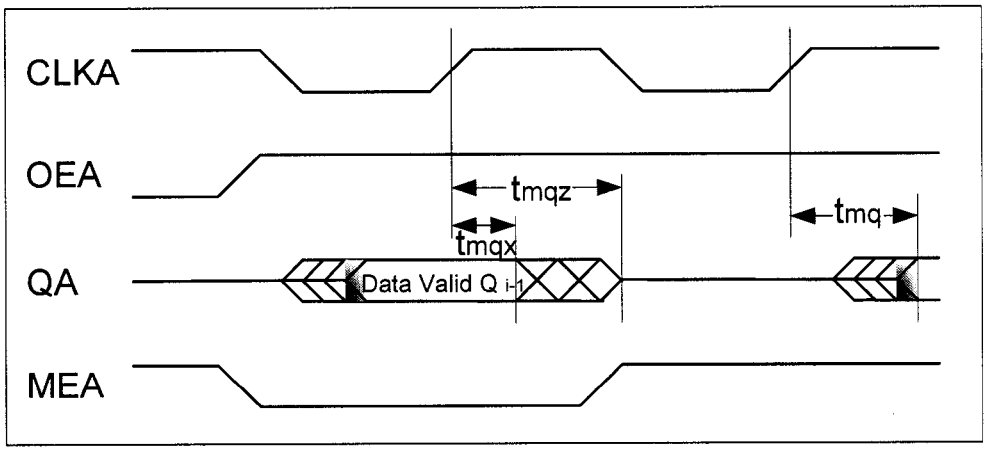
Read Cycle Timing Waveform



Write Cycle Timing Waveform



OE_ME controlled output timing



Note: The above waveforms are for the case where all control signals are active High.

Note: OEA is MEA dependent (OEA will control QA only if MEA is active)

Power Dissipation

PVT corner	Worst	Typical	Best
Static Idd (μA)	6.527	0.977	0.634

Typical Power cycle

PVT corner	Worst		Typical		Best	
Description	RD	WR	RD	WR	RD	WR
Power Dissipation (μW/MHz) per port	7.281	8.009	9.180	10.098	12.108	13.319
Current (μA/MHz) per port	4.495	4.944	5.100	5.610	6.115	6.727

Worst Power cycle

PVT corner	Worst		Typical		Best	
Description	RD	WR	RD	WR	RD	WR
Power Dissipation (μW/MHz) per port	25.262	27.788	31.849	35.034	42.007	46.208
Current (μA/MHz) per port	15.594	17.153	17.694	19.463	21.216	23.337

1. Static Idd is due to junction and sub-threshold leakage current (approximated).

2. Typical Power Cycle
   - Alternate cycles are active
   - 1/2 Address bits switching
   - 1/2 Data bits switching
   - Average of read 1/read 0
   - Average of address high/low

3. Worst Power Cycle
    - Back to back cycles
    - All address bits switching
    - All data bits switching
    - Worst of read 1/read 0
    - Worst of address high/low

Timing Name Convention

ADR : Address input
D : Data input
OE : Output Enable pin
ME : Memory Enable pin
CLK : Clock input pin
Q : Output

H : Logic High
L : Logic Low
Z : Float
X : No longer a valid logic level