TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Optimization of Cell-Aware Test

Citation for published version (APA):
Gao, Z. (2023). *Optimization of Cell-Aware Test*. [Phd Thesis 1 (Research TU/e / Graduation TU/e), Electrical Engineering]. Eindhoven University of Technology.

Document status and date:
Published: 10/03/2023

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 05. Oct. 2023

Optimization of Cell-Aware Test

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Eindhoven, op gezag van de
rector magnificus prof.dr.ir. F.P.T. Baaijens, voor een
commissie aangewezen door het College voor
Promoties, in het openbaar te verdedigen
op vrijdag 10 maart 2023 om 16:00 uur

door

Zhan Gao

geboren te Henan, China

Dit proefschrift is goedgekeurd door de promotoren en de samenstelling van de promotiecommissie is als volgt:

voorzitter:	prof.dr. E. Lomonova
1^e promotor:	prof.dr. K.G.W. Goossens
2^e promotor:	prof.dr.ir. S. Hamdioui (Technische Universiteit Delft)
leden:	prof.dr. A. Singh (Auburn University)
	prof.dr.ir. P.G.M. Baltus
	dr.ir. P.J.A. Harpe
	ir. E.J. Marinissen (IMEC)

Het onderzoek dat in dit proefschrift wordt beschreven is uitgevoerd in overeenstemming met de TU/e Gedragscode Wetenschapsbeoefening.

Optimization of Cell-Aware Test

Zhan Gao

Committee:

prof.dr. K.G.W. Goossens	Eindhoven University of Technology, *promotor*
prof.dr.ir. S. Hamdioui	Delft University of Technology, *second promotor*
prof.dr. A. Singh	Auburn University
prof.dr.ir. P.G.M. Baltus	Eindhoven University of Technology
dr.ir. P.J.A. Harpe	Eindhoven University of Technology
ir. E.J. Marinissen	IMEC

© Zhan Gao 2023. All rights are reserved. Reproduction in whole or in part is prohibited without the prior written consent of the copyright owner.

This thesis is typeset in LaTeX. The figures are created with Microsoft PowerPoint and the graphs are plotted using Microsoft Excel.

Printed by Proefschrift Maken – The Netherlands.

A catalogue record is available from the Eindhoven University of Technology Library.
ISBN: 978-90-386-5658-8

Abstract

The logic part in an integrated circuit (IC) is typically tested by various automatic test pattern generation (ATPG) approaches. Cell-aware test (CAT) is a new ATPG-based approach for logic circuits. It explicitly targets defects inside library cells and therefore significantly increases test quality compared to conventional ATPG-based approaches that cover cell-internal defects only serendipitously. Even though CAT's methodology has been defined and developed for more than ten years, we still identified and addressed several problems which helped to optimize the CAT results. The IC test has three key performance indicators (KPIs): test quality, test cost, and test development time, in order of decreasing importance. Therefore, this thesis focuses on optimizing the three test KPIs for CAT.

CAT consists of two phases: (1) library characterization and (2) cell-aware ATPG. In Phase 1, library characterization, potential cell-internal defects are identified for each library cell. Afterwards, a dedicated analog simulation is performed to determine which cell-level test pattern detects which cell-internal defect. This information is encoded in a defect detection matrix (DDM) per library cell. In Phase 2, cell-aware ATPG, chip-level test patterns are generated per circuit design by expanding the cell-level test patterns to chip level, according to the information in DDMs. DDMs, as the interface between both CAT phases, affect the CAT KPIs. We use three elements of DDMs: cell-internal defects, cell-level test patterns, and the detection result per defect-pattern tuple to set specific goals and propose corresponding solutions, with the aim to optimize the CAT

KPIs.

The particular set of cell-internal defects under consideration affects both test quality and test development time. Too few defects mean the test might be missing defects; this can negatively impact the test quality. Too many defects imply unnecessary time-consuming analog simulations during the library characterization, which increases test development time. Therefore, we propose an optimized library characterization flow that not only identifies all possible cell-internal defects to guarantee the test quality, but also reduces the library characterization time. The experimental results on an GPDK045 45nm library show that for 476 library cells the new library characterization flow saves 97.3% defect simulation time compared to simulating all potential defects.

For an intra-cell defect, only if a cell-aware ATPG tool expands at least one cell-level test pattern that detects this defect to a circuit level successfully, the defect is marked as 'covered' for the circuit. Cell-level test patterns, as the starting points of the cell-aware ATPG, influence the cell-aware ATPG results with respect to fault coverage, test pattern count, and compute time. Fault coverage is an indicator of test quality; test pattern count affects test cost; and ATPG compute time is a part of test development time. We propose two algorithms that manipulate cell-level test patterns in DDMs to optimize cell-aware ATPG results. Experimental results using the two algorithms in conjunction on twelve circuits show average reductions of 43% of non-covered faults and 10% in chip-pattern count compared to the ATPG results which are based on the original DDMs.

The fact of defect simulation is that in most defect-pattern tuples, the defect cannot be detected by the pattern. Based on our theoretical analysis, many undetectable tuples can be identified only on the basis of the topology of the cell's transistors netlist. We refer to this as logical undetectability. We propose an efficient algorithm that performs a structural analysis of library cells to identify all logically undetectable tuples, which then can be excluded from the time-consuming analog simulation. As our structural analysis is a lot faster than the analog simulation, their combination delivers significant speed-ups. For 476 standard cells from Cadence's GPDK045 library, we achieve a 47% reduction in overall characterization time.

CAT targets intra-cell resistive open and short defects. In case of a low-ohmic short or a high-ohmic open, we classify the defect as *hard*. Variants of the same defect with higher (for shorts) or lower (for opens) resistance exhibit a less impactful fault behavior than hard defects and hence are called *weak* defects. CAT library characterization is typically done for only two defect resistance values: one representing hard opens and another one representing hard shorts. To improve test quality, our goal is not only detecting hard defects, but also detecting weak defects. Similar to fishermen tightening the mesh size of their nets to catch small fish, we perform library characterization as efficiently as possible for a set of resistances representing increasingly weaker defects, and then adjust

our ATPG flow to explicitly target faults caused by the weakest still-detectable variant of each potential defect. To assess the effectiveness of our approach, we formulate a new dedicated test metric: the weakest fault coverage (WFC). Compared to conventional CAT targeting hard defects only, experimental results show that our new approach enhances the detection of weakest faults and significantly reduces WFC escapes, i.e., reduces (1 - WFC), while maintaining its original (hard-defect) fault coverage. This comes at the expense of (acceptable) increases in the required number of test patterns and associated test generation time.

We evaluate the effectiveness and efficiency of our optimization approaches on an advanced 3nm FinFET technology library and also the circuits based on this library. Combining with the optimized CAT results on Cadence's 45nm planar technology, we show that our optimized CAT tool flow works on both planar and FinFET CMOS technologies. We also compare the optimization results between the 45nm and 3nm technology library cells. The two libraries took comparable library characterization time as their numbers of potential intra-cell defects have a marginal difference.

All optimization methods and experiments in this thesis were implemented using the state-of-the-art CAT tool flow that was developed by Cadence Design Systems with our collaboration. The original results of this CAT tool flow are used as the baseline for comparisons with our experimental results. Three of our algorithms are a part of the Cadence commercial tool and are being used by some top-ten semiconductor companies.

Samenvatting

Het logische deel van een geïntegreerde schakeling (IC) wordt typisch getest not verschillende benaderingen voor het genereren van automatische testpatronen (ATPG). Zogenaamde cell-aware test (CAT) is een nieuwe ATPG-gebaseerde benadering voor logische circuits. Het richt zich expliciet op defecten in bibliotheekcellen en verhoogt daarom de testkwaliteit aanzienlijk in vergelijking met conventionele op ATPG gebaseerde benaderingen die cel-interne defecten slechts bij toeval afdekken. Hoewel de methodologie van CAT al meer dan tien jaar bestaat en wordt ontwikkeld, hebben we verschillende problemen geïdentificeerd en aangepakt die hebben bijgedragen aan het optimaliseren van de CAT resultaten. De IC-test heeft drie key performance indicators (KPI's): testkwaliteit, testkosten en testontwikkelingstijd, in volgorde van afnemend belang. Dit proefschrift richt zich op het optimaliseren van de drie test-KPI's voor CAT.

CAT bestaat uit twee fasen: (1) bibliotheekkarakterisering en (2) zogenaamde cell-aware ATPG (CA-ATPG). In fase 1, bibliotheekkarakterisering, worden mogelijke celinterne defecten voor elke bibliotheekcel geïdentificeerd. Daarna wordt een specifieke analoge simulatie uitgevoerd om te bepalen welk testpatroon op celniveau welk celintern defect detecteert. Deze informatie wordt gecodeerd in een defectdetectiematrix (DDM) per bibliotheekcel. In fase 2, CA-ATPG, worden testpatronen op chipniveau gegenereerd door de testpatronen op celniveau uit te breiden tot chipniveau, volgens de defectpatroondetectie-informatie in DDM's. Dit gebeurt per circuitontwerp. DDM's, als interface tussen fase 1 en 2, hebben invloed op de CAT KPI's. We gebruiken drie ele-

menten van DDM's: celinterne defecten, testpatronen op celniveau en het detectieresultaat per defectpatroon-tupel om specifieke doelen te stellen en bijbehorende oplossingen voor te stellen, met als doel de CAT KPI's te optimaliseren.

De specifieke verzameling cel-interne defecten die wordt overwogen beïnvloedt zowel de testkwaliteit als de testontwikkelingstijd. Te weinig defecten betekent dat de test mogelijk defecten mist en dit kan de testkwaliteit negatief beïnvloeden. Te veel defecten impliceert onnodige tijdrovende analoge simulaties tijdens de karakterisering van de bibliotheek, waardoor de testontwikkelingstijd toeneemt. Daarom stellen we een geoptimaliseerde karakteriseringsmethode voor de bibliotheek voor die niet alleen alle mogelijke celinterne defecten identificeert om de testkwaliteit te garanderen, maar ook de karakteriseringstijd van de bibliotheek verkort. De experimentele resultaten op een GPDK045 45nm-bibliotheek laten zien dat voor 476 bibliotheekcellen de nieuwe bibliotheekkarakteriseringsmethode 97,3% van de defectsimulatietijd bespaart in vergelijking met het simuleren van alle potentiële defecten.

Alleen als CA-ATPG bij een intraceldefect ten minste één testpatroon dat dit defect met succes op celniveau detecteert uitbreidt tot circuitniveau, wordt het defect gemarkeerd als "afgedekt" voor het circuit. Testpatronen op celniveau, als uitgangspunt van de CA-ATPG, beïnvloeden de CA-ATPG resultaten met betrekking tot foutdekking, aantal testpatronen en rekentijd. Foutdekking is een indicator van testkwaliteit; het aantal testpatronen beïnvloedt de testkosten; en ATPG-rekentijd is een onderdeel van de testontwikkelingstijd. We stellen twee algoritmen voor die testpatronen op celniveau in DDM's manipuleren om CA-ATPG resultaten te optimaliseren. Experimentele resultaten van de twee algoritmen, in combinatie, op twaalf circuits laten een gemiddelde reductie zien van 43% in niet-afgedekte fouten en 10% in het aantal chippatronen in vergelijking met de ATPG-resultaten die zijn gebaseerd op de originele DDM's.

Voor de meeste defect-patroon tupels in een DDM kan het defect niet worden gedetecteerd door het patroon. Op basis van onze theoretische analyse kunnen veel niet-detecteerbare tupels worden geïdentificeerd op basis van de topologie van de celtransistorsnetlijst alleen. We noemen dit logische niet-detecteerbaarheid. We stellen een efficiënt algoritme voor dat een structurele analyse van bibliotheekcellen uitvoert om alle logisch niet-detecteerbare tupels te identificeren, die vervolgens kunnen worden uitgesloten van de tijdrovende analoge simulatie. Omdat onze structurele analyse een stuk sneller is dan de analoge simulatie, levert dit een aanzienlijke tijdbesparing op. Voor 476 standaardcellen uit Cadence's GPDK045-bibliotheek bereiken we een karakteriseringstijdvermindering van 47%.

CAT richt zich op intracel resistante "open" (verbroken) en kortsluiting defecten. Bij een laagohmige kortsluiting of een hoogohmige open classificeren we het defect als *hard*. Varianten van hetzelfde defect met een hogere (voor kortsluiting) of lagere (voor

open) weerstand geven een kleinere fout dan harde defecten en worden daarom *zwakke* defecten genoemd. Karakterisering van de CAT-bibliotheek wordt meestal gedaan voor slechts twee defectweerstandswaarden: één die harde open defecten vertegenwoordigt en een andere die harde kortsluitingen vertegenwoordigt. Om de testkwaliteit te verbeteren, is ons doel niet alleen het detecteren van harde defecten, maar ook het detecteren van zwakke defecten. Net als vissers die de mazen van hun netten kleiner maken om kleine vissen te vangen, voeren we de bibliotheekkarakterisering zo efficiënt mogelijk uit voor een reeks weerstanden die steeds zwakkere defecten vertegenwoordigen. We passen vervolgens onze ATPG-toolflow aan om te focusseren op fouten die worden veroorzaakt door de zwakste nog detecteerbare variant van elk mogelijk defect. Om de effectiviteit van onze aanpak te beoordelen, formuleren we een nieuwe testmaatstaf: de zwakstefoutdekking (WFC). Vergeleken met conventionele CAT die alleen op harde defecten is gericht, laten experimentele resultaten zien dat onze nieuwe aanpak de detectie van de zwakste fouten verbetert en de WFC aanzienlijk vermindert, terwijl de oorspronkelijke (harde-defecten) foutdekking niet verandert. Dit gaat ten koste van een (aanvaardbare) verhoging van het benodigde aantal testpatronen en bijbehorende testgeneratietijd.

We evalueren de effectiviteit en efficiëntie van onze optimalisaties op een geavanceerde 3nm FinFET-technologiebibliotheek en de circuits die op deze bibliotheek zijn gebaseerd. In combinatie met de geoptimaliseerde CAT-resultaten op Cadence's 45nm planaire technologie, laten we zien dat onze geoptimaliseerde CAT-toolflow werkt op zowel planaire als FinFET CMOS-technologieën. We vergelijken ook de optimalisatieresultaten tussen de 45nm en 3nm technologiebibliotheekcellen. De twee bibliotheken hebben een vergelijkbare karakteriseringstijd van de bibliotheek, aangezien hun aantal potentiële intraceldefecten vergelijkbaar is.

Alle optimalisatiemethoden en experimenten in dit proefschrift zijn geïmplementeerd met behulp van de state-of-the-art CAT-toolflow die is ontwikkeld door Cadence Design Systems met onze samenwerking. De originele resultaten van deze CAT-toolflow worden gebruikt als basis voor vergelijkingen met onze experimentele resultaten. Drie van onze algoritmen maken nu reeds deel uit van de commerciële tools van Cadence en worden gebruikt door enkele top-tien halfgeleiderbedrijven.

Acknowledgments

Obtaining a PhD degree is a very challenging journey with many ups and downs for me. I stayed at Eindhoven University of Technology for the first and a half years and then moved to imec, Leuven for the rest of my PhD time. At both places, I was very fortunate to meet many great people supporting me to complete this journey. Without you, I would never finish it.

First, I must thank **Kees Goossens**, my first promoter, for giving me the opportunity to start the PhD research journey. I respect you from the depths of my heart, since you always do great in both work and life. In work, you are very responsible and efficient to fulfill your roles: lecturer, mentor, researcher, professor, and manager. Even though I had heard that you were a tough professor before I started my PhD, I think you always do the right things after I know you. You are also a kindhearted professor. I still remember you were worried about my mental health when I just came to the Netherlands from China. But you took care of me in a very considerate way: you asked a kind HR lady to have a regular talking with me, in case that I needed some help with a psychological problem. However, I had no mental problem actually, but just did not know how to communicate with a very professional professor, you, and our team members. I was too nervous to talk with people in English at the beginning of my PhD. In life, you have many good characters, such as your elegant English accent and fashion taste, that I am trying to imitate. No matter when, you are always present with a stylish dress code that enhances your professional image.

I am extremely grateful to my supervisor **Erik Jan Marinissen**. No words can express my gratitude to you. At any time when I think about you, endless stories pop up in front of my eyes. One week is not enough for me to tell these wonderful stories about working with you. You showed me that teaching by action is far more effective than by words. By observing how you work, it is no surprise that you have got such a good reputation in the IC test field. Your personality is too good to be true. Being responsible, being patient, being passionate about both work and life, and being warmhearted are just the tip of your personality iceberg. You have countless good characters, and I cannot believe that I could not find any shortcomings about you in the last five years. I bet that you will be my idol for the rest of my life, and I will keep pursuing to be a person like you. Here, I must mention **Adelia Maria de Oliveira Dias**, Erik Jan's wife, who so generously took time out of your schedules to drive for Erik Jan and me whenever we need. For instance, you drove to help me to move from Eindhoven to Leuven in 2018 and drove many times from Leuven to Eindhoven just for sending Erik Jan and me to have meetings there. You are also a good painter. Every postcard which you gave to me has beautiful painting and handwriting by you. Erik Jan and Adelia always remember my birthday and prepare presents for me. How lucky I am to encounter such a nice couple who take care of me like parents in Europe!

I also wish to thank **Jos Huisken** and **Hailong Jiao**, my daily supervisors at TU/e. Jos, you seem always happy. Your optimistic character always makes me relaxed to discuss with you and feel free to express my thought. You are a super technical man for me, being able to fix all things related to coding, software tools, and Unix. Hailong helped me to adapt to the European environment. When I just came to Europe from China, the environment changed too drastically for me and I was too shy to talk with anyone but you. You gave me much valuable advice not only on work but also on life. Even though we had worked together for only five months before you went back to China, you left a lot of good memories for me.

This endeavor would not have been possible without **Joe Swenton**, who is from Cadence Design Systems and worked together with me for the entire PhD journey. We had a lot of interesting and inspiring discussions in our weekly meetings. You also offered me the latest exciting new features in the tool underdevelopment. I will never forget our wonderful collaboration time. The interesting fact is that we had online meetings never with a camera on. I think I could easily recognize you through your voice but not your face in a crowd.

Special thanks to **Santosh Malagi**, **Min-Chun Hu**, and **Francesco Lorenzelli**, who did your master graduate projects with me. Each of you contributes to a chapter in this thesis. Santosh is so good at operating EDA tools and effectively generating and organizing data. Min-Chun is skillful at coding, while Francesco is mentally active and full of ideas. I am so happy that you three accompanied me in different phases of this

PhD journey. Due to your impressive capability shown in this PhD project, all of you achieved your plans for the career after the master study. Santosh joined Cadence Design Systems, US, and continued to be responsible for this PhD project from the Cadence side. Min-Chun also joined Cadence Design Systems and works for the Taiwan branch. Francesco managed to apply for a PhD candidate position at imec. Hope all of you have a bright future. I would like to also thank the promoters of these three master students, **Said Hamdioui**, **Cheng-Wen Wu**, and **Cecilia Metra**, for your support.

I gratefully acknowledge the assistance of Cadence Design Systems that offered me an exclusive platform to implement my PhD project and a financial support on travelling to academic conferences. In particular, I very much appreciate **Maxime Barbe**, **Vivek Chickermane**, **Franck Gerome**, **Walter Hartong**, **Anton Klotz**, **Luca Mattii**, **Robert McGowan**, **Wolfgang Meyer**, **Michel Montanuy**, **Sylvie Parmantier**, **Giuliano Sisto**, **Zhenzhou Sun**, **Mike Vachon**, **Carl Wisnesky**, and **Vladimir Zivkovic**, for your support. Many thanks to China Scholarship Council (CSC), which financially supported my living as a PhD student.

I would like to extend my sincere thanks to imec that has the most advanced semiconductor technology in this world and gave me access to it. At imec, I met many great colleagues who offered me help unconditionally. In particular, I would like to thank **Rogier Baert**, **Bilal Chehab**, **Julien Ryckaert**, and **Pieter Schuddinck** for contributing my conference paper published at IEEE International Test Conference, 2019. Moreover, special thanks to **Kristof Croes** and **Michele Stucchi** for being great team leaders by making me feel I really matter for the team. This gives me a lot of passion for work. I would like to also acknowledge the help of **Ingrid De Wolf**, **Dragomir Milojevic**, **Peter Debacker**, **Alessio Spessot**, **Ibrahim Tatar**, and **Diederik Verkest**.

I very much appreciate my leaders **Torstein Nesje** and **Linda Kristoffersen** in Nordic Semiconductor, who had sufficient patience with me. Thanks to their understanding, I was able to finish my PhD thesis by cutting 20% work time for Nordic Semiconductor in the first four months.

Thanks should also go to my defense chair **Elena Lomonova**, and committee members **Said Hamdioui**, **Adit Singh**, **Peter Baltus**, and **Pieter Harpe**, who helped me complete this journey. You were an integral part of me being able to make it to the end.

Due to my one and a half years experience in Eindhoven, and three years experience in Leuven, I was so lucky to encounter great people in both places who brighten my personal life. Thanks to my office mates at Eindhoven University of Technology (TU/e), **Paul Detterer** and **Kamlesh Singh**, I am not lonely to work overtime. I cannot count how many weekends we went to KFC to reward ourselves with fried chicken after hard work. Even though you both are talented and skillful, you still keep learning and being

humble. This makes me admire you so much. Thanks to **Alessandro Frigerio** and **Ilde Lorato**, I learned how to enjoy life in Europe. Except working, the rest of my weekend time became never boring. By the way, I like all your pasta recipes. **Savvas Sioutas**, thank you for teaching me English and correcting my English pronunciation. You are a really good English teacher. I would like to extend my sincere thanks to all my colleagues in the Electronics System Group. Thank you for your kind support. When I was living in Eindhoven, I need to travel to Leuven every week for having meetings with Erik Jan. Special thanks to **Jan Stuijt**, who was willing to offer me a weekly ride. You also told me many wonderful stories about your life during the driving and finally, we became good friends. I am also grateful to my Chinese friends in Eindhoven. In particular, thank **Yonghui Li**, **Xianhong Ouyang**, **Haijing Li**, **Jinlin Han**, and **Xiaoran Liu**. In Leuven, I received the first warm help from the imec colleagues **Fu-Xiang Liang**, **Kuba Kaczmarek**, **Leonidas Katselas**, **Myriam Van De Peer**, **Ferenc Fodor**, **Alicia Lesniewska**, and **Rafal Magdziak**. Afterwards, I had great pleasure of being friends with **Lizhou Wu**, **Huizi You**, **Yangyin Chen**, **Jing Li**, **Qi Lin**, **Yunxin Liu**, **Si Sun**, **Yusi Chen**, **Simei Yang**, **Qiuqang Lin**, and **Zheyi Li**.

Last but certainly not least, I would like to extend my sincere thanks to my family. I met my boyfriend, **Xiaoyu Piao**, at imec cafeteria, and we fell in love with each other at first sight. You accompany me to get through my PhD journey. In countless days, you wait for me to get off work until late at night and delivery food when I work at imec during weekends. You eliminate my loneliness and let me know how wonderful it is to have a partner. Special thanks to my parents, **Guoyan Gao** and **Yingcun Wang**, for always pushing me to become better and standing behind me with endless love.

Zhan Gao
Leuven, January 9, 2023

Contents

Abstract iii

Acknowledgments ix

1 Introduction 1
- 1.1 Logic IC Test . . . 3
  - 1.1.1 Basis of Automatically Generated Logic IC Test . . . 3
  - 1.1.2 Key Performance Indicators for Logic Test . . . 5
- 1.2 Overview of Cell-Aware Test . . . 8
  - 1.2.1 Cell-Aware Test Methodology . . . 8
  - 1.2.2 Key Performance Indicators for Cell-Aware Test . . . 10
- 1.3 Problem Statement . . . 11
- 1.4 Proposed Approaches and Thesis Organization . . . 14
- 1.5 Thesis Contributions . . . 15
  - 1.5.1 Industry Impact . . . 15
  - 1.5.2 Scientific Contributions . . . 16

2 Cell-Aware Test Fundamentals 17
- 2.1 Automatic Test Pattern Generation . . . 17
- 2.2 Related Prior Work . . . 24
- 2.3 Cadence's CAT Tool Flow . . . 26
- 2.4 Defect-Detection Matrix . . . 27
- 2.5 Defect Location Identification . . . 31
  - 2.5.1 Parasitic Extraction For Defect-Location Identification . . . 31

2.5.2 Full-Set of Defect Location Identification 38
2.6 Defect Simulation 41
2.6.1 One-Cycle Pattern Detection 44
2.6.2 Two-Cycle Pattern Detection 44
2.7 Cell-Aware ATPG 46
2.8 Conclusion 53

3 Defect Collapsing to Reduce Library Characterization Time 55
3.1 Proposed Library Characterization Flow 56
3.2 Open-Defect Identification and Simulation 57
3.2.1 Open-Defect Equivalence and Collapsing 57
3.2.2 Open-Defect Simulation and Equivalence Verification 65
3.3 Short-Defect Identification and Simulation 68
3.3.1 Short-Defect Equivalence and Collapsing 68
3.3.2 Short-Defect Simulation and Equivalence Verification 73
3.4 Experimental Results 77
3.4.1 Overview of GPDK045 CMOS Library 77
3.4.2 Experimental Results of Example Cell AND2X1 77
3.4.3 Experimental Results of GPDK045 Library 82
3.5 Conclusion 87

4 Optimization of Cell-Aware ATPG Results by Manipulating DDMs 89
4.1 Don't-Care Bit Identification Algorithm 90
4.2 MINCOVER Algorithms 94
4.3 Experimental Results 100
4.4 Conclusion 108

5 Speeding up Library Characterization by Preceding Simulation with Structural Analysis 109
5.1 Modeling of Library Cell Structure 110
5.2 Defect Detectability Analysis 112
5.3 Undetectability Checker Algorithm 115
5.4 Experimental Results on GPDK045 Library 117
5.5 Conclusion 120

6 Optimization of Cell-Aware Weak-Fault Coverage 121
6.1 Related Prior Work 125
6.2 Library Characterization 126
6.3 Library Characterization Results 130
6.3.1 Simulation of One-Cycle Cell Patterns 130
6.3.2 Simulation of Two-Cycle Cell Patterns 133
6.4 Weakest-Fault Cell-Aware ATPG 134

6.5 ATPG Experimental Results 135
6.6 Conclusion 138

7 Application of Cell-Aware Test on an Advanced 3nm Technology Node 139
7.1 iN5 Standard-Cell Library 140
7.2 iN5 Library Characterization 145
7.3 GPDK045 Versus iN5 149
7.4 iN5 Cell-Aware ATPG 151
7.4.1 ATPG Comparison: Stuck-at vs. Gate-Exhaustive vs. Cell-Aware 152
7.4.2 CA ATPG Optimization 156
7.4.3 Weakest-Fault Coverage Optimization 158
7.5 Conclusion 159

8 Conclusion and Future Work 161
8.1 Conclusions 161
8.2 Future Work 163

Publication List 165

About the Author 176

CHAPTER 1

Introduction

Since the first working silicon transistor was developed at Bell Labs in 1954, the semiconductor industry has been growing rapidly. Our daily lives have been drastically changed by integrated circuits (IC) in the last several decades. Nowadays, we are surrounded by electronic products, such as desktop and notebook computers, mobile phones, tablets, and smart home applications. Besides, new types of applications, like self-driving cars and internet of things are emerging. The “brains" of these products are ICs offering various complex and intelligent functions.

Figure 1.1 shows a generic IC product creation flow. Based on marketing requirements and an intensive interaction between customers and developers, required functionalities

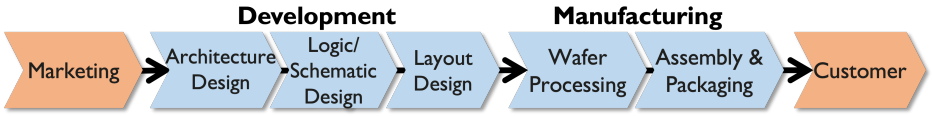


Figure 1.1: IC product creation process.

and architecture of an IC must be clearly defined and documented for the development process that follows. During the development process, the circuit architecture and behavior are described in the first step. Secondly, for digital circuits, designers use a logic synthesis tool to turn the desired circuit behavior into logic gates, while for an analog circuit, designers draw a transistor schematic to implement the circuit. This step is called logic or schematic design. In the third step, layout design is performed and then handed over to the manufacturer. At each development step, verification is required to make sure the circuit always behaves as expected.

In the manufacturing process, ICs are built upon wafers which are thin slices of semiconductors. In the wafer processing step, each wafer undergoes many fabrication processes, such as doping, annealing, thin-film deposition of various materials, lithography, and etching. A wafer can contain hundreds of duplicated dies. In the assembly and packaging step, the individual dies are separated by wafer dicing and packaged as chips.

Due to their large number of high-precision manufacturing steps, ICs are susceptible to fabrication defects that can cause circuit malfunctions. Therefore, chip yield, the percentage of chips that function correctly, is almost impossible to be 100%. As a specific manufacturing defect might occur in one single chip only, *all* chips need to be tested individually to weed out the defective parts before they are shipped to customers. The principle of IC testing is that we apply test stimuli at the circuit inputs and check if the corresponding test responses at circuit outputs are correct. If yes, the circuit passes the test; otherwise, the circuit fails. The combination of a stimulus set which specifies each circuit input value and the corresponding correct responses is defined as a *test pattern*. An IC can contain multiple types of module, such as logic circuit, embedded third-party intellectual property (IP) core, embedded memory, analog circuit, and radio frequency (RF) circuit. Various test methods were invented and developed for each type of module to generate test patterns. However, the test metrics are universal for all test approaches. IC testing has three universal key performance indicators (KPIs): (1) test quality, (2) test cost, and (3) test development time.

Among various logic test pattern generation approaches, cell-aware test (CAT) has an advantage of improving the test quality. Several leading semiconductor companies, such as AMD, Infineon Technologies, and Intel, published CAT results for volume testing [Hapke et al., 2012, Hapke et al., 2014a, Howell et al., 2018]. All the results showed that the CAT outperformed the traditional logic test approaches: stuck-at test [Eldred, 1959], transition-delay test [Waicukauski et al., 1987], and IDDQ test [Gulati and Charles, 1993]. In this thesis, we propose various approaches to optimize CAT in terms of the three test KPIs. The implementation of the optimization work requires a CAT tool flow reliable and flexible, such that we can embed our algorithms into the flow and verify the algorithms' effectiveness. Due to this requirement, we collaborated with Cadence Design Systems, which enabled us to access the Cadence's CAT tool development environ-

ment and tested our approaches there. After our research results were proven, Cadence adopted our approaches for the tool versions to be released, which results in an efficient release of new versions of the CAT tool.

The remainder of this chapter is organized as follows. Section 1.1 introduces the basics of automatically generated logic test and identify specific key performance indicators (KPIs) for logic test based on the universal IC test KPIs. In Section 1.2, we describe the CAT methodology and derive CAT KPIs compared to traditional logic test KPIs. In Section 1.3, we first set the goal of optimizing the CAT KPIs as the motivation of this thesis; subsequently, some unaddressed problems are stated. We outline our proposed solutions and the organization of this thesis in Section 1.4. In Section 1.5, we list the contributions of this thesis with respect to the state-of-the-art in the CAT research development.

1.1 Logic IC Test

1.1.1 Basis of Automatically Generated Logic IC Test

For logic circuits, the test pattern generation was automated by electronic design automation (EDA) technology. To introduce the logic test pattern generation flow, we should first explain the definitions of defect, fault, and fault model. Various types of non-perfect manufacturing, such as conductive particles between two wires, over-etching of wires, and half-filled contacts, called *defects* can happen anywhere in a logic circuit. It is not practical to generate test patterns for all physical defects individually. Fortunately, multiple types of physical defects show the same anomalous behavior, which is defined as a *fault*. Detecting a fault can be equivalent to detecting multiple physical defects. Therefore, targeting faults makes testing practical and efficient. The faults in digital ICs are classified into *fault models* which define fault types and locations in a circuit.

A basic logic test generation and operation flow is shown in Figure 1.2. It consists of three steps. In Step 1, we insert design for testability (DfT) hardware, scan chains, into the circuit gate-level netlist which is synthesized based on the register-transfer level (RTL) design and a pre-designed standard-cell library. The gate-level netlist with the DfT hardware is built up from interconnected standard cells. In Step 2, an automatic test pattern generation (ATPG) tool is used to generate test patterns for the circuit, based on the fault models selected by the ATPG tool users. In Step 3, the generated test patterns are applied on every individual fabricated chip and defective ones are rejected.

Today's industrial digital circuits can easily contain millions of standard cells but generally have a quite limited number of inputs. Through the limited number of circuit inputs

and outputs, it is difficult to apply stimuli at each circuit-inside fault location to activate the fault and to propagate the fault behavior to the circuit outputs. Therefore, the main challenges of testing a fault inside the circuit are *controllability* and *observability*. Controllability means that we should be able to set up a circuit to activate a fault if it is present, and observability requires to set up a circuit to observe a fault. Scan design addresses these problems by replacing all functional flip-flop (FF) cells in the circuit by scan flip-flop (SFF) cells and connecting all SFF cells into one or multiple chains [Williams and Angell, 1973]. The difference between FF and SFF cells is shown in Figure 1.3. An SFF cell contains a two-to-one multiplexer cell and an ordinary FF cell, such that an SFF cell has two inputs: one input is the regular input `D` and the second input is scan input `SI`. Selection between `D` and `SI` is done by scan enable `SE`. The scan output `SO` is tapped off regular output `Q`.

An example scan chain is shown in Figure 1.4. All SFF cells are daisy-chained by concatenating the output `SO` of an SFF to the input `SI` of another SFF. We connect the `SI` of the first SFF in a chain to a chip input pin `SI`$'$. The `SO` of the last SFF is connected to a chip output pin `SO`$'$. Scan enable and clock pins of all SFFs are controlled by the same chip input pin `SE`$'$ and the same clock signal `CLK`, respectively. In the scan-enabled mode (i.e. `SE`$'$ = 1), we obtain full controllability and observability of the values in the SFFs. During testing, test stimuli are first shifted into the SFFs under the scan-enabled mode. Subsequently, the circuit is set to scan-disabled mode (i.e. `SE`$'$ = 0), namely

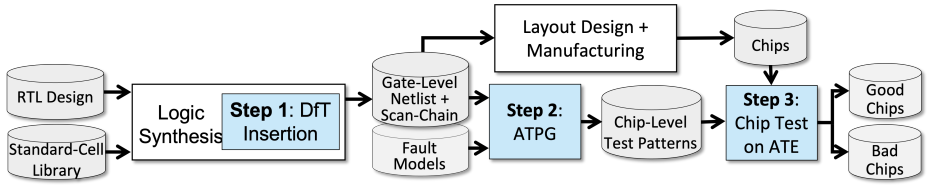
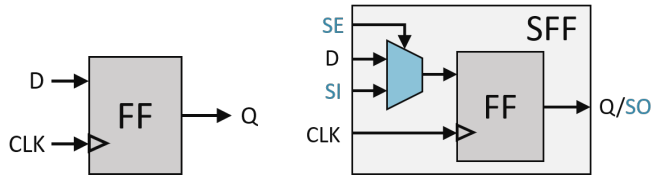


Figure 1.2: Logic IC test generation and operation flow.



(a) Flip-flop cell. (b) Scan flip-flop cell.

Figure 1.3: Comparison of flip-flop and conceptual scan flip-flop cells.

functional mode. After one clock cycle, the responses of the combinational part are captured into the SFFs. In the end, we change the circuit to scan-enabled mode (i.e. $SE' = 1$) again and shift the responses to the chip outputs. A scan chain makes every SFF work as both a virtual input and a virtual output pin of the combinational logic. All SFFs, primary inputs (PIs), and primary outputs (POs) constitute controllable and observable ports of a circuit. Scan-based design enables structural testing with enhanced controllability and observability for a circuit.

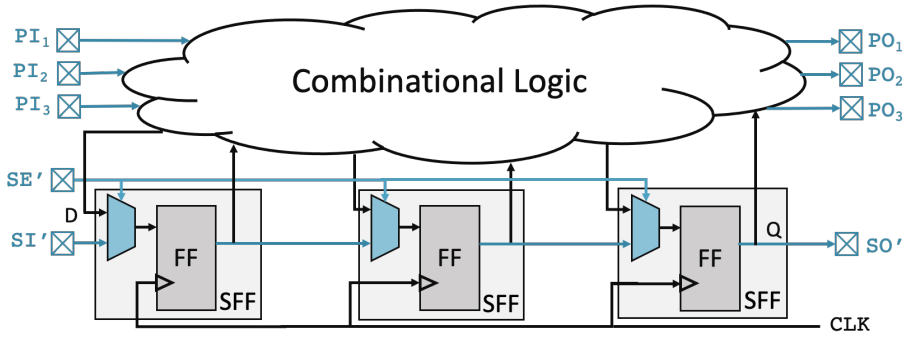


Figure 1.4: Example of a three-SFF scan chain.

Given a circuit gate-level netlist with the scan structure and a specific fault model, an ATPG tool automatically generates test patterns with the aim to detect all faults in the fault model for the circuit. *Fault coverage*, the percentage of detected faults out of all faults, denotes the ATPG effectiveness. The function of an ATPG algorithm is, given a fault, to apply a stimulus to activate the fault and then to expand the stimulus and the activated fault effects to circuit inputs and outputs, respectively. The combination of generated circuit input and output signals is a *test pattern*.

1.1.2 Key Performance Indicators for Logic Test

IC testing has three key performance indicators (KPIs): (1) test quality, (2) test cost, and (3) test development time, which are universal to all testings for all types of IC modules. In this section, we explain the three KPIs one by one and map them onto specific metrics of logic test.

Test quality. The objective of IC testing is to weed out all defective parts which cause chip malfunctions, before chips are shipped to customers. However, testing is not perfect, i.e., some defective parts can pass the test and cause *test escapes*, because the

applied test stimuli cannot detect those defects. Test escapes are typically quantified in defective parts per million (DPPM). The tolerated DPPM level is based on the product type. For example, the chip products used in safety-critical applications, such as healthcare and aerospace applications, target zero DPPM, while chips in consumer electronics can tolerate $\leqslant$100 DPPM. Test escapes will cause customers to return the defective products and seriously impact the reputation of the product brand. Therefore, all companies are striving to minimize the test escapes.

In the logic test, the Williams-Brown Equation [Williams and Brown, 1981] describes the relationship between test escape rate, yield, and fault coverage, shown as Equation 1.1, where *TER*, *Y*, and *FC* are test escape rate, circuit yield, and fault coverage, respectively.

$$TER = 1 - Y^{(1-FC)} \tag{1.1}$$

TER is inversely proportional to both *Y* and *FC*. Testing engineers cannot control *Y* which is the manufacturers' responsibility. Therefore, given a certain *Y*, improving *FC* causes *TER* to be reduced. *FC* or $(1 - FC)$ is an indicator of *TER* and test quality. In this thesis, to discuss test quality, we always look at $(1 - FC)$, because what really matters is how many faults are *not* covered. We call $(1 - FC)$ *fault coverage escape*. To compare two fault coverages, the reduction of fault coverage escape is an indicator of the *TER* reduction. For example, 99.0% and 99.5% are both high fault coverages. If we compare $(1 - FC)$, *FC*=99.5% reduces the escaped faults by half compared to *FC*=99.0%, which indicates a significant decrease of *TER*. Assuming 1 million logic circuits are manufactured with 80% yield, according to Equation 1.1, we calculate that the test escape rate is 2229 DPPM with the original 99% fault coverage, while 99.5% fault coverage causes 1115 DPPM. Hence, the DPPM reduction is 50.0%.

Test cost. Tests are executed on a specific equipment called automatic test equipment (ATE), which utilizes the test patterns to distinguish between defect-free and defective circuits. Test cost consists of two components: (1) DfT hardware area and (2) test execution cost. The DfT hardware is necessary to improve the testability of circuits. However, the DfT hardware enlarges the chip area and also requires testing. This increases the test cost. Test execution cost comes from equipment depreciation, equipment maintenance, and operation cost. As ATEs are expensive, normally their prices are several million dollars. Bushnell and Agrawal showed that in the year 2000, an ATE price could be 4.3 million dollars. They computed the yearly execution cost of the ATE by assuming a linear depreciation over five years, an annual maintenance cost of 2% of the purchase price, and an annual \$0.5 million dollars operating cost. Therefor, the annual test execution cost is (4.3/5) + (2% $\times$ 4.3) + 0.5 = 1.4 million dollars [Bushnell and Agrawal, 2000]. On average, the test execution costs 0.045 dollar per second with the assumption

that the ATE works all day of the year. Bushnell and Agrawal also told that a single IC on that ATE could take 6 seconds. Assuming ten million such ICs are fabricated, we can calculate the total test execution cost, which is 0.045 $\times$ 6 $\times$ 10 million = 2.7 millions dollars, on that ATE. Given an ATE, to reduce the test execution cost, the only way is to reduce the test execution time per chip that equals the sum of application time of all test patterns.

In logic circuits, the DfT hardware is one or multiple scan chains. The test cost on DfT hardware is quantified by the percentage of chip area, while the test execution time *TET* is the product of one pattern application time on an ATE $t_{pattern}$ and the number of test patterns #*patterns*, shown as Equation 1.2. Therefore, we can either decrease $t_{pattern}$ or decrease #*patterns* to reduce the test execution time. $t_{pattern}$ is the sum of scan-in time $t_{scan-in}$, capture time $t_{capture}$, and scan-out time $t_{scan-out}$, shown as Equation 1.3. $t_{scan-in}$ is identical to $t_{scan-out}$, which is determined by the number of SFFs in a scan chain noted as #*SFFs* and the ATE scan clock frequency f_{scan}, shown as Equation 1.4. We can build multiple parallel short scan chains instead of one long scan chain to reduce #*SFFs*, and hence $t_{pattern}$ is reduced. Also, we can reduce the number of test patterns #*patterns* to reduce *TET* per chip.

$$TET = t_{pattern} \times \#patterns \tag{1.2}$$

$$t_{pattern} = t_{scan-in} + t_{capture} + t_{scan-out} \tag{1.3}$$

$$t_{scan-in} = t_{scan-out} = \#SFFs/f_{scan} \tag{1.4}$$

Test development time. For each circuit, the test development includes the DfT hardware design and test pattern generation. Both of them require verification before and after layout to make sure the circuit functions properly prior to manufacturing. The test development time grows with the complexity of circuit architecture, and can be several months or more than one year. This is an non-recurring engineering (NRE) cost, as the test development is done only once per circuit design. We want a fast test development process to reduce time-to-market. Please note, after 22nm technology node, the chip development cost increases quickly with the technology scaling, shown in Figure 1.5. According to a survey from the International Business Strategy Corporation (IBS), IC design costs jumped from \$70.3 (1$\times$) million for a 22nm chip to \$297.8 (4.2$\times$) million for a 7nm chip, and \$542.2 (7.7$\times$) million for a 5nm chip. Lapedus reported that in the following 3nm technology node, a complex GPU development at Nvidia cost \$1.5 (21.3$\times$) billion [Lapedus, 2018]. The test development is a part of chip design, therefore, reducing the test development time is even more important for the chips based on the advanced technology nodes. Per logic circuit, test development time is the sum of scan-chain insertion time, scan-chain verification time, ATPG compute time, and the test pattern verification time.

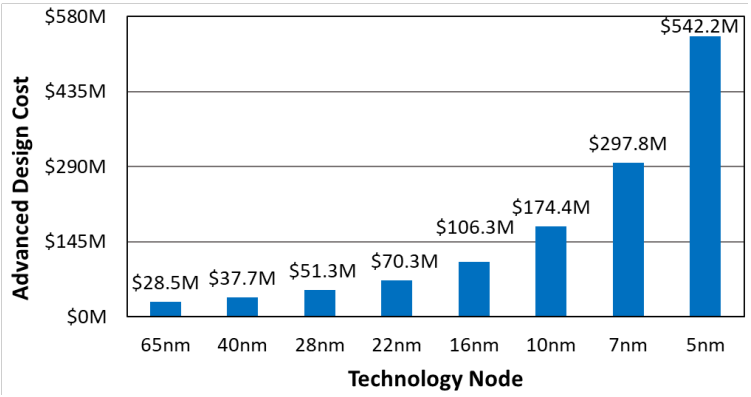


Figure 1.5: IC design cost per technology node (source: IBS).

We aim to optimize each of the test KPIs, in the decreasing order of importance: test quality, test cost, and test development time.

1.2 Overview of Cell-Aware Test

Conventional scan-based logic test methods, such as stuck-at test [Eldred, 1959], transition-delay test [Waicukauski et al., 1987], and bridging fault test [Ferguson and Larrabee, 1991], have been used in the semiconductor industry for several decades. In 2008, Eichenberger et al. [Eichenberger et al., 2008] showed, based on volume diagnosis, that a large fraction of test escapes resulted from cell-internal defects not covered by the conventional logic test approaches. Cell-aware test (CAT) was proposed in 2009, and since then, has shown its effectiveness on improving test quality by explicitly targeting intra-cell defects, with affordable test pattern count and test development time [Hapke and Schlöffel, 2012, Hapke et al., 2014a, Howell et al., 2018].

1.2.1 Cell-Aware Test Methodology

The CAT methodology consists of two phases, shown in Figure 1.6. In Phase 1, *library characterization* is carried out for each library cell. Analog simulation is performed to identify which cell-internal defect is detected by which cell-level test pattern. The defect simulation results are encoded, serving as input for Phase 2, *cell-aware ATPG*. For each

circuit design built up from interconnected library cells that are characterized in Phase 1, the ATPG tool tries to expand the patterns from cell to chip level with the aim to detect all the cell-internal defects. In this thesis, we use “cell pattern" as a shorthand for cell-level test patterns, and “chip patterns" or “test patterns" for chip-level test patterns.

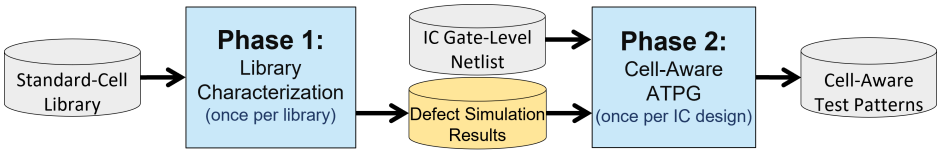


Figure 1.6: CAT methodology consists of two phases.

The defect simulation results can be encoded in various ways, but should contain three elements: defects, cell patterns, and detection result per defect-pattern tuple. We use a two-dimensional binary matrix, called defect detection matrix (DDM), to encode the simulation detection results per library cell. Figure 1.7 shows an example DDM for library cell AND2X1 from Cadence’s GPDK045 45nm library [Cadence Design Systems, 2014]. Defects and cell patterns are represented by columns and rows, respectively. Therefore, the number of columns equals the number of characterized defects, and each column header corresponds to a possible defect in the cell. The detection result per defect-pattern tuple (i.e. the DDM entry) is a boolean value. If and only if defect d is detected by cell pattern p, we specify the DDM entry logic 1 at the cross point of column d and row p.

Cell: AND2X1																										
One-Cycle Detection																										
Patterns	Short Defects													Open Defects												
AB/Y	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10	d11	d12	d13	d14	d15	d16	d17	d18	d19	d20	d21	d22	d23	d24	d25	d26
p0 00/0	1		1				1						1	1												
p1 01/0	1		1	1	1							1		1	1								1			
p2 10/0	1		1				1										1									
p3 11/1	1	1		1		1	1		1	1			1		1		1		1							
Two-Cycle Detection																										
p0 R1/R								1								1				1					1	
p1 F1/F											1									1		1		1		
p2 1R/R																1				1						1
p3 1F/F																						1		1		1

Figure 1.7: An example of defect detection matrix.

Each DDM column represents which cell patterns detect the corresponding defect. If no cell pattern detects a defect, this defect does not cause a fault and hence is removed from

the DDM for the downstream cell-aware (CA) ATPG. During CA ATPG, successful expansion of a cell pattern to chip level is not guaranteed and depends on the environment of the circuit-under-test (CUT). A defect may be detected by multiple cell patterns. To detect this defect, it is sufficient for the ATPG tool to expand only one of the cell patterns successfully. All cell patterns are kept for each defect to provide alternatives for the ATPG engine, in case some of the patterns cannot be expanded successfully in a circuit. The reasons of failed expansion will be explained in Section 2.1. The defects detectable by at least one cell pattern cause *faults* that are targets of ATPG.

Compared to the conventional logic test approaches which target faults at cell inputs and outputs (I/Os) only, CAT has an extra phase, library characterization, which causes CAT to be more effective and more efficient. The effectiveness comes from explicitly targeting all defects potentially happening inside cell which includes the defects at cell I/Os. Therefore, the CAT fault set per cell typically is a superset of the faults targeted by the conventional test approaches. The efficiency is improved because we identify the cell patterns specifically detecting these cell-internal defects, such that we do not include non-effective patterns.

1.2.2 Key Performance Indicators for Cell-Aware Test

To obtain the best CAT quality, we should first consider a comprehensive set of intra-cell defects for each library cell. Under this condition, in the cell-aware ATPG phase, the test quality is represented by the fault coverage. Based on Equation 1.1, we can reduce the test escape rate by increasing the fault coverage.

Identical to the conventional ATPG methods, the CAT cost consists of scan-chain area and test execution time. CAT requires scan chains the same with the conventional scan-based logic test approaches. However, CAT generates more test patterns as a cost for the higher test quality than the conventional ATPG approaches. This causes longer test execution time.

Conventional ATPG-based test and CAT development time includes scan-chain insertion time, scan-chain verification time, ATPG compute time, and the test pattern verification time. Compared to the conventional ATPG approaches, CAT requires additional development time which is the library characterization time. It is an NRE cost per cell library, but a time consuming work as the defect simulation process consists of three nested loops: for each library cell, every intra-cell defect should be simulated with the exhaustive set of cell patterns individually. For example, the library characterization for Cadence GPDK045 library which contains 476 library cells took ~ 81 days. A specific technology library can be updated and released by foundries several times, which means

in reality the library characterization may be preformed several times. Therefore, it is essential to reduce the library characterization time. In the three nested loops, the number of cells in a library cannot be reduced, therefore we can either reduce the number of intra-cell defects or the number of cell patterns for each defect to decrease the simulation time.

In summary, the effectiveness of simulated intra-cell defects and the CA fault coverage affect the CAT quality. The inserted scan-chain area and the number of chip patterns influence the CAT cost. The library characterization time, scan-chain insertion time, and ATPG compute time constitute the CAT development time.

1.3 Problem Statement

Our goal is to improve the CAT KPIs, specifically:

1. Improving test quality.
   1.1 Maximizing the modeling of realistic intra-cell defects.
   1.2 Maximizing the CA fault coverage.
2. Reducing test cost.
   2.1 Reducing the chip-level test pattern count.
3. Reducing test development time.
   3.1 Reducing the library characterization time.
   3.2 Reducing the CA ATPG compute time.

Based on the five sub-goals, we propose and address five problems in this thesis. As the three DDM elements (intra-defects, cell patterns, and detection result per defect-pattern tuple) clearly influence the CAT KPIs, we use them as vehicles to propose the problems. The relationship among KPIs, goals, problems, and DDM elements is shown in Figure 1.8. The first two problems are proposed based on intra-cell defects, the third problem based on cell patterns, and the last two problems based on defect-pattern tuple.

The detailed problem descriptions are as follows.

P1 **Identify a comprehensive set of realistic defect locations.** Intra-cell defects, as one of the DDM elements, affect test quality. A defect is defined by its type, location, and resistance. CAT targets two types of defects: resistive shorts and opens. These defects can potentially happen anywhere with different probabilities. The set of locations for potential defects should be realistic and complete. Too few defect locations mean the test might be missing defects; this can negatively impact test quality. Too many

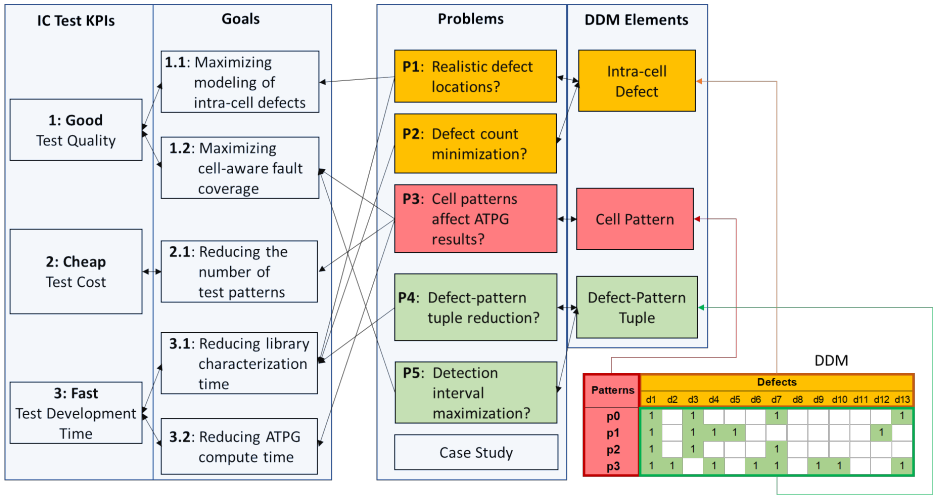


Figure 1.8: Relationship among KPIs, goals, problems, and DDM elements.

defect locations imply unnecessary time-consuming analog simulations during the library characterization, and hence negatively impact the test development time.

P2 **While identifying a comprehensive set of potential defects, can we reduce the number of defects to be simulated, with the aim to reduce the library characterization time?** The fault caused by a defect is detected by a set of cell patterns which are determined by the library characterization. Various defects can be detected by the same cell patterns, i.e. cause the same fault. If we can group the defects that are detected by the same set of cell patterns before the defect simulation, per defect group we only need to simulate one representative defect. Thus, the number of defects is reduced and the simulation time decreases.

P3 **Can cell patterns be optimized to improve the CA ATPG results?** Cell patterns contain logic signals to be expanded to the circuit level by the ATPG engine. Therefore, they affect the ATPG results: fault coverage, test pattern count, and ATPG compute time. In the DDMs originally generated from the CAT library characterization, all bits of each cell pattern are specified by 0 or 1. Each specified bit is an expansion task for the ATPG engine. The question is if all the specified cell-input values are necessary to detect a defect? If a defect is still detectable by a cell pattern with some non-specified bits, the expansion of this partly-specified cell pattern is easier for the ATPG engine. These non-specified bits are called *don't-care* bits. Exploiting don't-care bits helps to reduce the pattern expansion complexity for the ATPG tool. Besides, given a library cell, do all cell patterns require to be expanded to cover all cell-internal defects during the CA ATPG? The effectiveness of different cell patterns

can be different. For a cell, can prioritizing the expansion of cell patterns that detect more defects first accelerate the ATPG process for this cell? If we can accelerate the ATPG for all cells in a circuit, the ATPG compute time will be reduced for this circuit.

P4 **Is it possible to reduce the library characterization time by pruning the defect-pattern tuples to be simulated?** The library characterization time is the sum of defect-location identification and defect characterization time. The latter is dominant. In the original CAT defect characterization, all defect-pattern tuples are simulated individually. With the increase of cell structure, the number of cell patterns and the number of defects both increase, which causes the number of defect-pattern tuples increases drastically. For example, a multi-bits flip-flop with seven inputs and 50 transistors from Cadence's GPDK045 library requires a simulation for $\sim$210,000 defect-pattern tuples, and the total simulation time by Cadence's CAT tool is $\sim$4 hours. For a library containing more than 1000 cells, it is critical to reduce the defect simulation time for the CAT development.

P5 **Target the maximum detection interval per defect location.** An open or short defect is modeled by a linear resistor with a certain value. The defect resistance value is selected based on the severity of the corresponding physical defect. In the case of a high-ohmic open or a low-ohmic short, we classify the defect as 'hard'; if the defect is less severe and its resistor value starts to slide (to lower values for open defects or higher values for short defects), the defect is classified as 'weaker'. Weaker defects exhibit a less impactful fault behavior than hard defects and therefore are often more challenging to detect. At each defect location, not only a hard but also a weak defect could happen. The commercial CAT tools from Cadence Design Systems and Mentor Graphics use a single short and a single open defect resistance, normally two hard defect resistances, for all short and open defect locations, respectively. We can model multiple defect variants by changing the resistance value of a defect. Per defect-pattern tuple (d, p), with the d resistance varying from hard to weak, d becomes more and more difficult to be detected by p. Finally, at a resistance value, d becomes non-detectable by p. This resistance value represents the weakest defect detectable by p. Thus, each defect-pattern tuple (d, p) has a weakest defect resistance value. By successfully expanding p, we cover all defect variants harder than the weakest variant detectable by p. Among all cell patterns detecting defect d, if we successfully expand the cell pattern that detects the weakest defect, we cover the largest detectable interval at d location. Thus, we improve the test quality.

1.4 Proposed Approaches and Thesis Organization

Figure 1.9 shows a high-level overview of this thesis organization. In Chapter 2, we present a CAT tool flow as the baseline of all optimization approaches proposed in this thesis. This CAT tool flow is created in collaboration with Cadence Design Systems and was commercially released in 2019. Furthermore, we introduce the detailed basis of CAT, and solve Problem 1: identifying a complete set of defect locations.

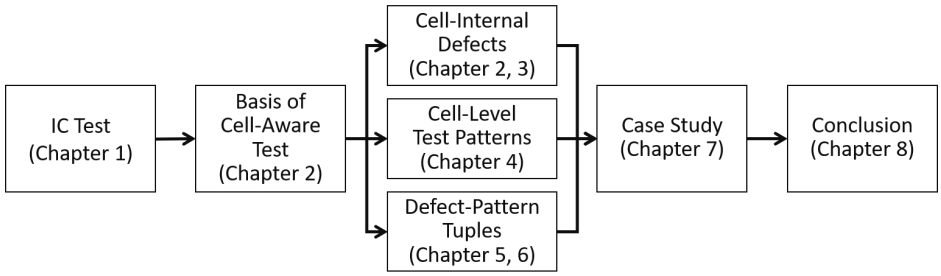


Figure 1.9: Overview of the organization of this thesis.

In Chapter 3, we solve Problem 2 by proposing a library characterization flow in which we minimize the number of simulated defects to reduce the simulation time without a negative impact on the test quality. The defects detected by the same set of cell patterns are called equivalent defects. After identifying a comprehensive set of defects, we identify equivalent defects and collapse every group of equivalent defects into one defect which is to be simulated. Afterwards, we perform two-round defect simulations. The first round defect simulation is used not only for identifying cell patterns for each defect, but also for verifying the accuracy of defect collapsing. Only if some non-equivalent defects are collapsed, we perform the second round simulation to add the missed faults caused by these non-equivalent defects for the downstream cell-aware ATPG.

In Chapter 4, Problem 3 is solved to optimize the CA ATPG results which influence all three test KPIs, by developing two DDM manipulation algorithms. One algorithm expands the set of full-specified cell patterns by identifying and adding partly-specified cell patterns with don't-care bits to offer more alternatives for the ATPG tool to expand. The other algorithm, referred as MINCOVER algorithm, determines the minimal set of cell patterns that cover all cell-internal defects. Thus, we can prioritize the most efficient set of cell patterns for the ATPG process with the aim to reduce the ATPG compute time.

To solve Problem 4, in Chapter 5, we first perform per defect-pattern tuple an analysis based on the cell structure to determine if the pattern potentially detects the defect. The definitely non-detectable defect-pattern tuples can be identified only on the basis of the topology of the cell's transistor netlist. We refer to this as logical undetectability. Afterwards, we present an efficient algorithm that performs a structural analysis of library cells to identify all logically undetectable defect-pattern tuples, which then can be excluded from the time-consuming analog simulation. As our structural analysis is a lot faster than the analog simulation, their combination delivers significant speed-ups.

In Chapter 6, we solve Problem 5. Per defect, we simulate a range of defect resistances from hard to weak for each defect. The analog simulation helps to identify, per defect-pattern tuple, the weakest version that is still detectable. Afterwards, a new methodology called weakest-fault ATPG is proposed to maximize both the weakest and hard fault coverages for circuit designs.

Chapters 3–6 give experiment results based on Cadence's 45nm planar CMOS technology library called GPDK045 [Cadence Design Systems, 2014]. This demonstrates the applicability of the solutions to planar CMOS technology. To show the robustness of the methodologies, we perform a case study in Chapter 7 for an advanced 3nm technology. As far as we know, we are the first to report CAT results for such an advanced technology. We present conclusions and future work in Chapter 8.

1.5 Thesis Contributions

1.5.1 Industry Impact

We collaborated with Cadence Design Systems to develop a CAT tool flow. This tool flow was first released as a commercial product in 2019 and is used as the baseline of the work in this thesis. We would like to highlight that our algorithms listed below were embedded into the released versions of the Cadence CAT tool, Modus, and are being used by several top ten semiconductor companies.

1. Full set of defect-location identification (DLI) algorithm in Chapter 2.
2. Defect-collapsing algorithm in Chapter 3.
3. MINCOVER algorithm in Chapter 4.

The remaining algorithms in this thesis are waiting to be embedded into the future versions. Besides, the dedicated PEX settings for DLI were explored by us with the assis-

tance of Cadence PEX tool department and were written into the Modus manual to guide users; the ASCII DDM format was designed by us and implemented in Modus.

1.5.2 Scientific Contributions

In this thesis, we made the following contributions to the state-of-the-art of the CAT methodology.

1. We used Parasitic EXtraction (PEX) to assist the cell layout analysis and identify a full set of defects locations with the aim to guarantee the cell-aware test quality.
2. Based on the equivalence of defect behaviors, we design a library characterization flow with simulating only a small fraction of defects from the full set. Thus, the defect simulation time, and hence the library characterization time was reduced. Our experiment shows that the defect simulation time for the GPDK045 45nm technology library is reduced by 97.3% compared to the simulation based on the full set of defects.
3. For Problem 3, we provided two DDM manipulation algorithms and three usages in the cell-aware ATPG to demonstrate the effectiveness of each algorithm on improving the fault coverage, on reducing chip-level test patterns, or on reducing the ATPG compute time. For a set of twelve benchmark circuits, on average the non-covered faults were reduced with 45% and the test pattern count was reduced with 10% using the algorithms in conjunction compared to the ATPG which is based on the original DDMs.
4. Cell structure analysis, the solution of Problem 4, offered a deep understanding of defect simulation, which helped (1) to reduce the simulation time by filtering out the logically undetectable defect-pattern tuples, (2) to explain why the weakest detectable defect resistance varies with the defect location, (3) to verify the simulation results. The simulation time reduction was 47% for the used GPDK045 library due to this analysis.
5. Weakest-fault CAT extended the CAT detection interval from one defect variant to a range of defect variants. For the twelve benchmark circuits, we detect the maximum defect-variant range for 99.4% of the defect locations.
6. We demonstrated the applicability of the thesis work on an advanced 3nm technology, which is the node the industry will get in the near future. We are also the first to prove the CAT methodology can handle the very advanced technologies.
7. Testability and defect sensitivity information of standard cells offered by our CAT flow can be used as an additional metric for library and circuit design on top of the traditional metrics, area, power, and performance.

CHAPTER 2

Cell-Aware Test Fundamentals

In this chapter, we introduce in Section 2.1 the automatic test pattern generation (ATPG) process to all fault models. ATPG-based test approaches and the CAT development history are presented in Section 2.2. In Section 2.3, we describe Cadence's CAT tool flow which is used as the baseline of this thesis work, and which we further develop in this thesis together with Cadence. Section 2.4 introduces the three DDM elements: defects, cell-level test patterns, and detection result per defect-pattern tuple. Section 2.5 discusses how we use parasitic extraction (PEX) to assist with identifying the cell-internal defect locations. Section 2.6 describes the defect simulation implementation in the Cadence CAT tool flow. Section 2.7 presents the CA ATPG process by comparing it to conventional ATPG approaches.

2.1 Automatic Test Pattern Generation

Given a fault model and the gate-level netlist of a circuit design, the ATPG tool tries to generate test patterns for this circuit that cover all the faults contained in the fault model.

The gate-level netlist of a logic circuit is built up from interconnected cell instances that are from a pre-designed cell library. Typically, these cells have a standard height which enables cells to be aligned in rows and share the power rails at the top and bottom of each row; therefore the library cells are called *standard cells*. The fault model tells the ATPG tool which faults to target. The ATPG tool generates test patterns through four basic steps.

Step 1 *Fault sensitization* determines the logic value at the fault location such that the target fault is exposed.

Step 2 *Fault propagation* sets logic values for the circuit downstream of the target fault to make the fault effect visible at some circuit output.

Step 3 *Fault justification* sets logic values for the circuit upstream until circuit inputs such that we can identify a circuit-level stimulus pattern that enables the fault sensitization and propagation.

Step 4 *Fault simulation* determines which other faults are covered by the test pattern generated from the first three steps.

A test pattern consists of a test stimulus pattern for all circuit inputs and a response pattern for all circuit outputs. For logic circuits, the stimulus and response values are either 0 or 1. To distinguish the stimulus and response bits, we use '0' or '1' to represent the stimulus value and use 'L' and 'H' to represent the logic low (0) respectively high (1) of the response value. For example, test pattern '01/L' implies we should apply the two stimulus bits '01' at the chip-under-test inputs and the response value should be logic 0.

The two most popular fault models used in industrial logic testing are the stuck-at [Eldred, 1959] and the transition delay [Waicukauski et al., 1987] fault model. The stuck-at fault model considers two static faults: a signal always stays at logic 0 or logic 1, and the stuck-at fault location is an input or an output (I/O). The I/O in a circuit can be a primary I/O or a standard-cell instance I/O. The transition-delay fault model considers two delay faults: signal on an I/O changes from 0 to 1 with a delay longer than the circuit functional clock cycle, called slow-to-rise fault; similarly, the other fault is slow-to-fall on an I/O. We first use the stuck-at fault model as a vehicle to illustrate the four-step ATPG process. Subsequently, with the basis of the stuck-at ATPG process, we introduce the transition-delay ATPG process. Figure 2.1 shows a small circuit with a stuck-at-0 fault on wire *d*. The ATPG process for the stuck-at-0 fault is as following.

Step 1 To sensitize this fault, we should set the logic value on *d* to be logic 1. If we set logic 0 on *d*, the fault cannot be observed. Roth [Roth, 1966] formalized

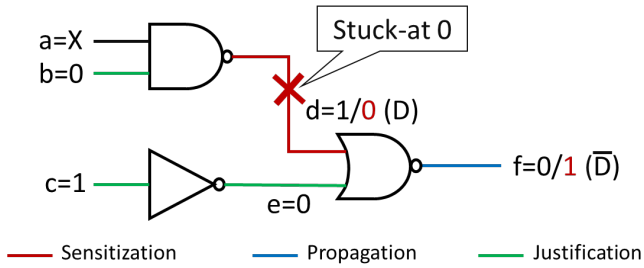


Figure 2.1: An example of the ATPG process for a stuck-at 0 fault.

the ATPG process with the D-Algorithm in which D represents a situation that the correct logic value should be 1, but the faulty value 0 is present. Similarly, $\overline{D}$ means the good value is 0, but the bad value 1 is present. By applying the D-Algorithm, the activated stuck-at-0 on d is marked by D in Figure 2.1.

Step 2 To observe the fault effect at output wire f, logic 0 should be assigned to wire e because logic 1 on e causes f to be 0, independent of the logic value on d, and the fault effect cannot be observed at output f. Thus, the correct output on f is logic 0 but due to the stuck-at-0 on d, the faulty output is logic 1. In the D-Algorithm, $\overline{D}$ is present on f.

Step 3 The signal values to be justified consists of two parts: (1) the signal sensitizing the fault and (2) the signals generated for the fault propagation. In Figure 2.1, to have logic 1 on d (signal sensitizing the fault) and logic 0 on e (signal propagating the fault), we need to assign at least one input of the NAND cell to be 0 and set 1 on input c, respectively. With input b set to be logic 0, independent of the logic value on a, d is logic 1, and the fault is sensitized. Therefore, it is unnecessary to assign a logic value to a for detecting the stuck-at 0 fault. We define *a don't-care* bit, denoted by 'X'. The generated test pattern is 'X01/H', where 'H' means that the correct response at the output f is logic 1. We use red, blue, and green colors to present the interconnects on which logic values are set for sensitization, propagation, and justification, respectively.

Table 2.1 explicitly shows propagation and justification tasks step by step. Before the ATPG starts, both of the propagation and justification task lists P and J contain only signal D on wire d and all bits of the test pattern is unidentified, shown as the row of Step 0. In Step 1, we propagate D from wire d to f by assigning logic 0 to e. Therefore, the propagation task is updated to (f, $\overline{D}$). This propagation step causes one more justification task (e, 0) which

is added to the justification task list. In Step 2, the ATPG tool recognizes wire f is the circuit output, so the propagation task is finished and updated to an empty set. The correct logic value on f is used to specify the response bit of the test pattern. Justification process for the tasks (d, D) and (e, 0) is shown in Steps 3–6. After (d, D) is justified to the gate, this task is replaced by (a, X) and (b, 0). As a and b are both circuit inputs, the justification for (d, D) terminates at the two wires and the stimuli bits for these two inputs are determined. The justification for (e, 0) identifies input c to be 1. After all justification tasks are finished, the test pattern is generated in Step 6.

Step No.	Propagation & Justification Tasks	Test Pattern *abc/f*
0	$P = \{(d, \mathrm{D})\}$ $J = \{(d, \mathrm{D})\}$	XXX/U
1	$P = \{(f, \overline{\mathrm{D}})\}$ $J = \{(d, \mathrm{D}),(e, 0)\}$	XXX/U
2	$P = \emptyset$ $J = \{(d, \mathrm{D}),(e, 0)\}$	XXX/L
3	$P = \emptyset$ $J = \{(a, \mathrm{X}),(b, 0),(e, 0)\}$	XXX/L
4	$P = \emptyset$ $J = \{(e, 0)\}$	X0X/L
5	$P = \emptyset$ $J = \{(c, 1)\}$	X0X/L
6	$P = \emptyset$ $J = \emptyset$	X01/L

Table 2.1: Propagation and justification task set change step by step.

Step 4 Fault simulation identifies that test pattern 'X01/H' also detects stuck-at-1 on wire f. In the D-Algorithm, all Ds and $\overline{\mathrm{D}}$s in the circuit denote the faults detected by the same pattern.

We now discuss the second fault model, transition-delay. Detecting a transition-delay fault on an I/O requires two clock cycles. From the first to the second clock cycle, we should apply a logic signal with a transition either from 0 to 1 or from 1 to 0. The transition happens at the time point when the second cycle starts. Due to the transition-delay fault, the logic value on the I/O does not change in the second cycle in presence of a transition-delay fault. Therefore, the fault effect is exposed in the second cycle, and is then equivalent to a stuck-at fault. Figure 2.2 shows the transition-delay fault effect. The green waveform represents a correct signal transition without any delay in the second

clock cycle, and the red waveform shows a transition with a delay fault which causes the transition to not be observed in the second cycle.

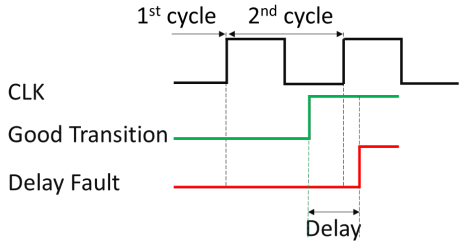
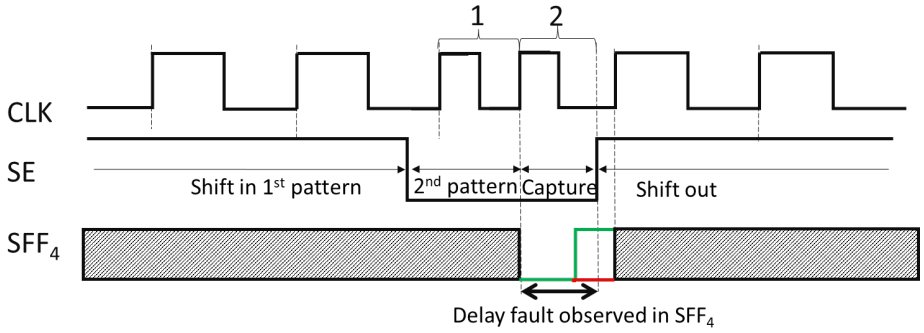


Figure 2.2: Transition-delay fault effect.

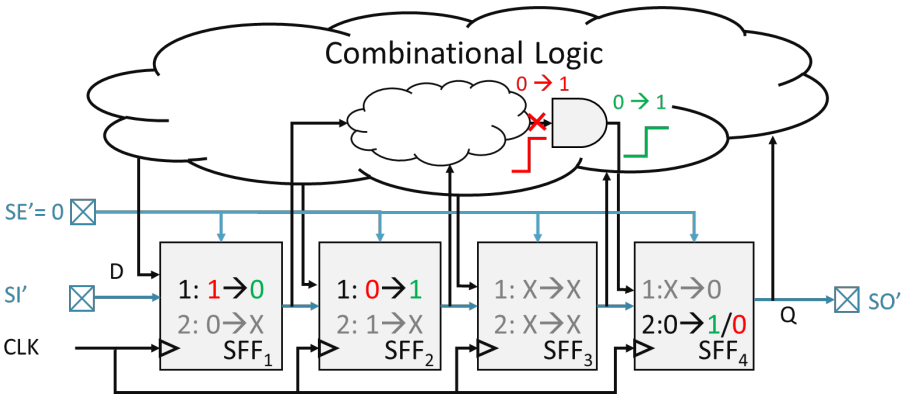
To generate the test pattern for a transition-delay fault, the ATPG tool first expands the initialization value on the I/O to circuit inputs to generate test stimuli for the first clock cycle, and then generates a second pattern to detect a stuck-at fault for the second clock cycle. The ATPG process for transition-delay faults should consider creating the transition with the scan design. The approach used in the industrial tests is called launch-on-capture (LOC) ATPG. The initial pattern is scanned into the scan chains one SFF by one SFF in the scan-enabled mode. Subsequently, we set to function-enabled mode. In the first functional clock cycle, the first pattern initializes the I/O with the target transition-delay fault, and the combinational-cell responses of the first pattern are pushed to the inputs of SFFs. When the second functional clock cycle comes, the responses of the first pattern are pushed into the SFFs and act as the stimuli of the second test pattern. In the second cycle, the target transition-delay fault is activated and propagated to a primary output or a SFF. Afterward, we set to scan-enabled mode again and scan out the responses from the scan chains. The challenge of the LOC ATPG is that the responses of the first pattern should be the stimuli of the second pattern. It is difficult to satisfy this condition in a large-scale or complex circuit, therefore the fault coverage of transition-delay test is typically low (i.e. $\sim$70-80%).

We use a slow-to-rise fault as an example to illustrate the launch-on-capture process, shown in Figure 2.3. The waveforms of the clock, the scan-enable (SE), and the SFF that work together to capture delay fault are drawn in Figure 2.3(a). During the scan-disabled period, the clock is faster than the scan-mode clock. Because the delay faults are time sensitive, the two-cycle test pattern should be applied with an at-speed clock. The slow-to-rise fault in Figure 2.3(b) is propagated to scan flip-flop SFF_4, therefore in Figure 2.3(a), we cannot observe a rise transition in SFF_4 during the capture period. Figure 2.3(b) shows the signals used to activate and propagate the slow-to-rise fault. The slow-to-rise fault requires a rise transition at the fault location. At the end of scan-in

mode, the bits in SFF_1 and SFF_2 initialize the fault location with a logic 0. In the first clock cycle during the scan-disabled mode, the values in SFF_1 and SFF_2 change from the initial stimuli to the responses to the initial stimuli, and cause a rise transition at the fault location. In the next clock cycle, the fault effect is propagated into SFF_4. Subsequently, we change to scan-disabled mode and shift out the responses in the SFFs. In each SFF of Figure 2.3(b), we fill the value changes, where 'X' is a don't-care bit, during the two clock cycles in the scan-disable mode. Therefore, the test pattern for this delay fault is '10XX/XXXH'.



(a) A slow-to-rise delay fault detection waveforms.



(b) Stimuli and responses in two clock cycles when scan-enable is disabled.

Figure 2.3: Two-cycle test pattern application and transition-delay fault detection.

The basic concept of ATPG is expanding logic values from the fault location to circuit inputs and outputs. For all gates in the expansion (i.e. propagation and justification)

path, the logic values set on the gates I/Os are determined by the logic gate functions. However, due to the highly complex structure of industry designs, the expansion cannot be always successful. It is possible that opposite logic values are required on the same wire during the expansion. This is called a *signal conflict*. Therefore, the fault coverage cannot always reach 100%. An example of signal conflict is shown in Figure 2.4. To detect the stuck-at-0 fault on wire d, we propagate it to the output g through wire e. During the fault justification, wire f should be assign to 1, which requires wires c and d both are logic 1. However, d cannot be logic 1, as it is stuck at 0. Typically, in a large circuit a signal can be propagated or justified through multiple paths. When the signal expansion encounters a signal conflict, the ATPG tool *backtracks* and tries another path. For example, in Figure 2.4 if the ATPG tool backtracks to wire d and propagates D to output g through wire f, e should be assigned 1. Fault justification derives $a = 1$, $b = 0$, and $c = 1$ without any signal conflict. If a signal cannot be expanded successfully through any path, we call the corresponding target fault *non-detectable*.

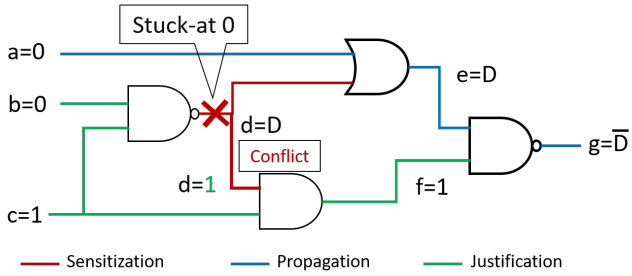


Figure 2.4: Signal conflict on wire d.

For a large logic circuit including millions of library cells, each generated test pattern contains ∼95% don't-care bits because the fault propagation and justification go through a quite limited number of cells and reach a limited number of circuit inputs and outputs. These don't-care bits have two usages: (1) *low-power test* and (2) *pattern compression*. Power consumption during the test execution should be considered, as each test pattern are scanned in and out of the scan chains bit by bit, which causes a lot of signal activities in both SFFs and combinational cells. High power consumption may cause the chip under test to overheat, and behave abnormally. This negatively impacts the test quality. If we make many bits in sequence identical, the number of signal transitions per SFF is reduced. For example, assuming an original test pattern is 'XXX0XXX1', we fill all don't-care bits by logic 0 to generate a test pattern '00000001'. During the eight-bit pattern '00000001' is scanned into a scan chain with eight SFFs, and each SFF flips only once. Thus, the power consumption is minimized. An example of pattern compression is as follows. Assuming two test patterns, '0X1/H' and 'X01/H', are generated, we can merge them into '001/H' by specifying their don't-care bits with the care bits in the other

pattern bit by bit. In this example, the first bit in ‘X01/H’ is ‘X’ and is specified by the first bit ‘0’ in ‘0X1/H’. The second bit ‘X’ of ‘0X1/H’ is specified by the second bit ‘0’ of ‘X01’. The third bit and the response bit of both patterns are identical. Therefore, we can compress the two patterns into one. Pattern compression reduces the number of test patterns, and hence reduces the test execution time. However, it causes more care bits in the test patterns, and may increase the power consumption during chip test.

2.2 Related Prior Work

In addition to the stuck-at and transition-delay fault models, other fault models, such as bridging [Ferguson and Larrabee, 1991], transistor stuck-open [Wadsack, 1978], and path delay [Smith, 1985] fault models, have been developed. However, these fault models either have insufficient test quality or are too expensive for production volume tests.

Stuck-at test is still generally accepted as the basis for industrial logic test due to the following advantages. (1) The stuck-at faults effectively model physical defects compared to other fault models [McCluskey and Tseng, 2000], which is demonstrated by industry testings. (2) The ATPG is only based on the gate-level netlist, which make it easy to implement. (3) The ATPG for SA faults has a high fault coverage. If it is difficult for an ATPG tool to get a high enough fault coverage based on a fault model, then the fault model is useless for ATPG-based logic testing.

Based on the stuck-at fault model, the following various approaches were proposed to improve the test quality of ATPG-based approaches. n-*detect* [Ma et al., 1995] tries to generate a test pattern set which covers each stuck-at fault n times with different test patterns (where $n \in \mathbb{N}$ and $n > 1$). The hope is that the additional test patterns (on top of regular (‘1-detect’) test patterns) which are generated in this way might increase the coverage of other, non-modeled non-stuck-at faults. Obviously, it is important that additional test patterns do not just repeat the original stuck-at test patterns; hence, crucial for an n-detect approach is the definition of *‘different’* for test patterns. The first n-detect paper [Ma et al., 1995] considers two patterns for one stuck-at fault to be different when they differ in at least one bit, even if this bit does not make a contribution to the detection of this stuck-at fault. Later, several papers try to improve on this definition, by requiring that two test patterns are considered different if and only if they activate or propagate the target fault in different ways [Pomeranz and Reddy, 2001] or show different logic values in the physical neighborhood of the fault site [Blanton et al., 2001]. Unfortunately, n-detect comes at the cost of an almost linear increase in test patterns [Venkataraman et al., 2004]. *Embedded multi-detect* (EMD) [Geuzebroek et al., 2007] exploits the many don’t-care bits of conventional (1-detect) stuck-at patterns to achieve multiple stuck-at

detections and therefore does not suffer from the expensive test pattern increase of n-detect. However, its exploitation of don't-care bits prevents those from being utilized for other purposes, such as test data compression and/or low-power test [Miyase et al., 2002]. Both n-detect and EMD detect non-stuck-at faults only serendipitously, and thus, their effect on test quality enhancement, although definitely presents, is unpredictable. [Eichenberger et al., 2008] shows that the stuck-at faults are insufficient to model all fault effects caused by intra-cell defects, hence not surprisingly, the non-covered cell-internal defects are found to be the root cause of a significant fraction of test escapes.

Gate-exhaustive test [Cho et al., 2005] improves test quality in a brute-force way by trying to apply *all* input combinations at library-cell level; this will undoubtedly guarantee full coverage of all cell-internal faults. However, this approach results in an unnecessarily high number of test patterns, especially for library cells with a large number of inputs. For example: an 8-to-1 multiplexer with in total 11 inputs will generate $2^{11} = 2048$ cell-level test patterns, many of which do not contribute to additional fault coverage. Furthermore, as GE test does not target any specific faults, it cannot help with diagnosis and failure analysis.

Inductive fault analysis [Shen et al., 1985] improves test quality by analyzing all physical defects per manufacturing process step with an assumption that unintended missing and extra materials occur at random locations. Subsequently, the physical defects are classified into different fault types based on their electrical behaviors. Analog simulation is then performed to identify the test patterns for the faults. [Shen et al., 1985, Ferguson and Shen, 1988b, Ferguson and Shen, 1988a] shows the better effectiveness of inductive fault analysis on small circuits, a full-adder and a 4×4 multiplier, compared to stuck-at test. However, inductive fault analysis is intractable for modern large-scale circuits containing billions of logic gates.

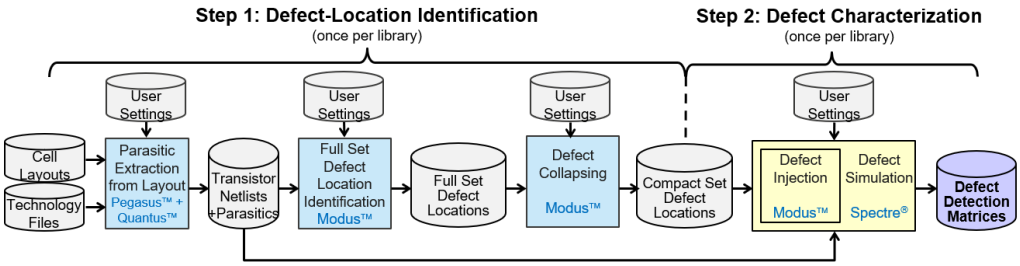
To improve test quality with affordable test development effort, CAT was inspired by inductive fault analysis, which is impractical to be applied at the chip level, but very feasible if applied on individual library cells which are the building blocks of logic circuits. CAT performs analog simulation at the cell level to identify cell patterns for the fault effects caused by intra-cell defects and uses the ATPG technique to generate chip-level test patterns.

CAT was introduced by Hapke et al. in 2009 [Hapke et al., 2009]. Initially, only hard (low ohmic) short defects (1Ω) in combinational logic cells were targeted. Later, this was extended to hard (high ohmic) open defects (1GΩ), weaker short defects (1Ω – 20kΩ) [Hapke et al., 2010, Hapke et al., 2011a], and sequential cells such as scan flip-flops [Hapke et al., 2014b]. Several studies have shown that CAT offers superior test quality and reduces test cost in comparison with n-detect [Ma et al., 1995, Pomeranz and Reddy, 2001, Hapke and Schlöffel, 2012], EMD [Geuzebroek et al., 2007, Zhang et al., 2014],

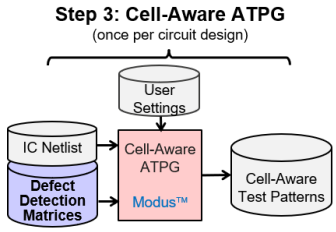
or GE test [Cho et al., 2005, Hapke et al., 2011b]. Industrial application has provided experimental evidence of the effectiveness of CAT to reduce test escapes [Hapke et al., 2011a, Hapke and Schlöffel, 2012, Yang et al., 2014, Sinha et al., 2017, Howell et al., 2018]. CAT also improves the diagnosis accuracy and efficiency by recording the layout location of potential defects [Maxwell et al., 2016, Tang et al., 2017].

2.3 Cadence's CAT Tool Flow

The Cadence CAT tool flow is depicted in Figure 2.3. It consists of three steps: (1) defect-location identification (DLI) [Gao et al., 2019b], (2) defect simulation, and (3) cell-aware ATPG. The first two steps constitute the CAT library characterization and are executed once per cell-library version; the last step is specific to a particular IC design. The interface between Steps 1 and 2 on the one, and Step 3, on the other hand, is the defect simulation results which are encoded in a *defect-detection matrix* (DDM) per library cell.



(a) Library characterization.



(b) Cell-aware ATPG.

Figure 2.5: Cadence's cell-aware test tool flow.

Step 1, *defect-location identification* [Gao et al., 2019b] first determines a full set of

defect locations with the assistance of parasitic extraction (PEX) [Prahov and Graupner, 2011]. The *full set* contains all potential locations of both open and short defects for intra-cell interconnections and transistors. Defects potentially happening on and between cell-internal interconnections are identified based on layout analysis implemented by Cadence's parasitic extraction (PEX) tool Quantus™. Transistor-internal defects are modeled as opens on and shorts between transistor terminals. Subsequently, the *defect collapsing* function in Cadence's ATPG tool Modus™ creates a *compact set* of defect locations out of the full set by selecting one representative defect location for each set of defects that have the same fault effect. This compact set with defect locations is typically significantly smaller than its corresponding full set. The reduction translates itself into a proportional reduction of simulation time in Step 2. The full set of defect locations is stored for later use during failure analysis.

Step 2, *defect simulation* first adds a resistive value to the defect locations. The tool allows user-defined defect resistances. Subsequently, the defects and the defect-free netlist are submitted to the *Transient Fault Analysis* function of Cadence's analog simulator Spectre® [Schaldenbrand et al., 2020]. For a given defect d and cell-pattern p, if and only if the simulation of corresponding defect-free and defective netlists result in a different cell output value, p is capable of detecting defect d, which is denoted by DDM(d, p) = 1. Per library cell, we generate a DDM described in the next section.

In Step 3, *cell-aware ATPG*, for a circuit design, an ATPG engine expands the patterns in DDMs from cell to chip level to detect as many cell-internal defects as possible and to minimize the number of chip patterns.

2.4 Defect-Detection Matrix

Figure 2.6 shows an example DDM in which we encode the defect simulation results. A DDM is a two-dimensional binary matrix and contains three elements: defect, cell-level test pattern, and detection result per defect-pattern tuple, which are discussed in this section.

We define DDMs for standard cells as follows. Let c be a standard cell with n inputs and m outputs, with $n, m \in \mathbb{N}^+$. Let D_c and P_c be the sets of defects respectively patterns for cell c.

Inspired by inductive fault analysis, CAT identifies locations of the defects in D_c by analyzing manufactured layers in every library cell. Instead of considering all physical defects like inductive fault analysis, CAT uses resistive open and short defects to model all physical defects. The reasons are as follows. First, it is widely accepted in the IC

Cell: ADDHX1																													
One-Cycle Detection for Static Faults																													
Patterns		Short Defects														Open Defects													
	AB/SC	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10	d11	d12	d13	d14	d15	d16	d17	d18	d19	d20	d21	d22	d23	d24	d25	d26		
p0	00/0-													1															
p1	01/1-										1			1															
p2	10/1-						1					1	1	1															
p3	11/0-						1				1	1		1															
p4	00/-0	1		1				1																					
p5	01/-0	1		1	1	1																							
p6	10/-0	1		1				1	1																				
p7	11/-1	1	1		1		1	1		1	1																		
Two-Cycle Detection for Delay Faults																													
p0	0R/R-										1			1												1			
p1	R0/R-						1					1	1	1									1		1				
p2	0F/F-													1													1		
p3	R1/F-						1				1	1		1									1		1		1		
p4	F0/F-													1													1		
p5	1R/F-						1				1	1		1						1		1					1		
p6	F1/R-										1			1												1			
p7	1F/R-						1					1	1	1									1						
p8	0R/-F	1		1	1	1																							
p9	R0/-F	1		1				1	1																				
p10	0F/-F	1		1				1																					
p11	R1/-R	1	1		1		1	1		1	1				1		1	1											
p12	F0/-F	1		1				1																					
p13	1R/-R	1	1		1		1	1		1	1				1		1	1											
p14	F1/-F	1		1	1	1																							
p15	1F/-F	1		1				1	1																				

Figure 2.6: An example defect detection matrix (DDM).

test community that manufacturing defects are accurately modeled as resistive shorts and opens [Bushnell and Agrawal, 2000]. Second, inductive fault analysis also demonstrated that the effects of physical defects are equivalent to opens on and shorts between interconnects or inside transistors [Ferguson and Shen, 1988a]. The defect resistance value reflects the defect size. For example, a large area of unintended metal connecting two interconnects causes a hard short defect which is modeled by a small resistance. If two interconnects are barely shorted by a small metal spot, the defect resistance should be large. In summary, a CAT defect is defined by location, type (i.e. open or short), and resistance value.

Cell patterns presented by rows in DDMs are the starting points of chip-level test pattern generation. The fault effects of an intra-cell defect are reflected on cell outputs and

propagated by the ATPG engine to circuit outputs. An intra-cell defect can cause an incorrect value on one or multiple cell outputs. To observe and detect a cell-internal defect at circuit level, it is sufficient to propagate any faulty response caused by this defect from the cell to circuit output. We use single-output cell patterns in DDMs. An example DDM for a half-adder cell is shown in Figure 2.6. The cell ADDHX1 has two outputs: 'S' and 'C'. Each cell pattern in the DDM gives a single bit response on either output S or C. For defect d impacting the m cell outputs when applying stimulus pattern *sp*, m cell patterns are recorded for d as alternatives for the ATPG process, each cell pattern consists of *sp* and each response of the *m* outputs.

A stimulus pattern can be fully-specified or partly-specified (containing one or more don't-care bits). In Figure 2.6, all cell patterns are fully-specified. The detection set of each partly-specified cell pattern can be computed from the detection of fully-specified cell patterns instead of running simulation. This will be presented in Chapter 4.

We use analog simulation to determine the cell malfunctions caused by the identified open and short defects. When applying stimuli, we consider two types of logic faults at cell outputs: static and delay faults. Static faults can be observed by applying specific static stimuli. Delay fault observation requires at least one cell-output transition, which requires specific input transitions. Therefore, each stimulus bit in a delay fault cell pattern can be a static or dynamic signal. A static signal is logic 0 or 1, and a dynamic signal is a transition from logic 0 to 1 or from logic 1 to 0. The rising transition is denoted by 'R', and the falling transition is denoted by 'F'. A cell pattern containing at least one dynamic stimulus bit implies two concatenated cell patterns with only static stimuli. For example, pattern 'R1/R' means we should sequentially apply patterns '01/0' and '11/1' in two test cycles. Therefore, cell patterns containing at least one dynamic stimulus bit are called *two-cycle* cell patterns. Similarly, cell patterns containing only static stimulus bits are called *one-cycle* cell patterns. As each stimulus bit can be 0, 1, R or F, the total number of fully-specified cell patterns is $m \times 4^n$ which is the sum of $m \times 2^n$ one-cycle cell patterns plus $m \times (4^n - 2^n)$ two-cycle cell patterns.

A DDM for cell c is specified by a function $DDM_c : D_c \times P_c \longrightarrow \{0,1\}$ with $DDM_c(d,p) = 1$ if and only if defect d is detected by pattern p. The *pattern set* for defect d is defined by function $ps_c : D_c \longrightarrow \mathcal{P}(P_c)$ with $ps_c(d) = \{p \in P_c |\ DDM_c(d,p) = 1\}$. Similarly, the *detection set* of pattern p is defined by function $ds_c : P_c \longrightarrow \mathcal{P}(D_c)$ with $ds_c(p) = \{d \in D_c | DDM_c(d,p) = 1\}$. In a DDM with $|P_c|$ rows, there can be at most $2^{|P_c|}$ different columns even though $|D_c|$ can be larger than $2^{|P_c|}$. If two defects d_1 and d_2 are detected by identical pattern sets, i.e. $ps_c(d_1) = ps_c(d_2)$, we call d_1 and d_2 *equivalent* defects. For the ATPG engine, a successful expansion of any pattern in $ps_c(d_1)$ detects both d_1 and d_2 for the circuit. Therefore, identical columns in DDM can be collapsed into one column for the ATPG process.

Defect d in cell c is called *non-detectable* if no element in the set of cell patterns detects d, i.e. $ps_c(d) = \emptyset$. Non-detectable defects are removed from D_c for the cell-aware ATPG, as ATPG has no means to detect them. The remaining defects are detectable at cell level and are referred to as *faults*.

```
****************************************************************
Cellname: AND2XL
****************************************************************

Defect Detection Matrix - one-cycle
-----------------------
Patterns         : 4
Candidate Defects: 20
Input pins: A, B, Output pins: Y
Sum | ID   | Pat     1                          10                  20
5   | p1   | 00 0  [ 1  0  0  0  0  0  0  1  0  1  0  0  1  0  1  0  0  0  0  0 ...
8   | p2   | 01 0  [ 0  0  0  0  1  1  0  1  0  1  0  0  1  1  1  1  0  0  1  0 ...
7   | p3   | 10 0  [ 1  1  0  0  0  0  0  0  1  0  1  0  1  0  0  1  0  1  0  0  0 ...
9   | p4   | 11 1  [ 1  0  1  1  1  0  1  1  1  0  1  0  1  0  0  0  0  0  0  0  0 ...
----------------------------------------------------------------------------------...
Sum                [ 3  1  1  1  2  1  1  4  1  3  1  1  3  1  3  1  1  0  1  0 ...


Defect Detection Matrix - two-cycle
-----------------------
Patterns         : 4
Candidate Defects: 20
Input pins: A, B, Output pins: Y
Sum | ID   | Pat        1                              10                  20
15  | p1   | 01:11 01 [ 1  0  1  1  1  0  1  1  1  0  1  0  1  0  0  0  0  0  1  0 ...
15  | p2   | 10:11 01 [ 1  0  1  1  1  0  1  1  1  0  1  0  1  0  0  0  0  0  1  0 ...
13  | p3   | 11:01 10 [ 1  1  0  0  0  0  0  1  0  1  0  1  0  0  1  0  1  0  0  0 ...
14  | p4   | 11:10 10 [ 0  0  0  0  1  1  0  1  0  1  0  0  1  1  1  1  0  0  0  0 ...
----------------------------------------------------------------------------------...
Sum                  [ 3  1  2  2  3  1  2  4  2  2  2  1  3  1  2  1  1  0  2  0 ...
```

Figure 2.7: An example DDM file printed by Modus based on our research.

We designed DDMs in ASCII format for Cadence's CAT tool flow. Figure 2.7 shows a DDM file for partial defects in the AND2X1 cell from Cadence's GPDK045 45nm technology library. In this file, the DDM is split into two matrices: one-cycle cell pattern DDM and two-cycle cell pattern DDM which require different ATPG runs to expand the cell patterns. In both DDMs, we first provide the defect and cell pattern counts and cell I/O pins. Subsequently, for the matrices, we also calculate each row and column sums. The row sum is the number of detected defects by the cell pattern in the row, and the column sum is the number of cell patterns that detect the defect in the column.

2.5 Defect Location Identification

The fundamental starting question for CAT implementation is how to identify defect locations. CAT targets open and short defects at various locations with different probabilities inside a library cell which consists of transistors and interconnects connecting transistors. Open defects occur in cell-internal interconnects or inside transistors. A long and/or thin interconnect has a high probability of an open defect. The shape of interconnect is reflected by its resistance. A long and/or thin wire leads to a high resistance. Short defects occur between cell-internal disjoint interconnects or inside transistors. If two wires run for a long distance in parallel and/or are close to each other, short defects are more likely to occur between such two wires. The probability of a short defect is related to the capacitive coupling strength between the two wires. The higher the capacitance extracted from a wire pair, the higher the probability of a short defect occurring between the two wires. Therefore, the resistance and capacitance for the interconnects are good indicators of defect occurring probability. The interconnect resistance and capacitance are called 'parasitics' in IC design because they are typically not part of the original design intent, but are present nevertheless and therefore need to be assessed by a parasitic extraction (PEX) tool to build an electrically accurate simulation model of the circuit in question. Therefore, we use the PEX tool to assist in the defect-location identification (DLI) for cell-internal interconnects connecting transistors. Based on literature search, we identify typical manufacturing defects inside metal–oxide–semiconductor field-effect transistors (MOSFETs), such as gate-oxide breakdown, improper doping, and poor gate etching. Related prior work shows that these MOSFET-inside defects can be modeled by either opens or shorts between transistor terminals. Each MOSFET has four terminals: gate, source, drain, and substrate, and hence has six terminal pairs. We summarize the prior work related to MOSFET-inside defects and identify three terminal opens and four terminal-pair shorts for a transistor with the aim to model all transistor-inside defects. This work is presented in Section 2.5.2.

In Section 2.5.1, we first introduce a cell model built based on the extracted netlists of library cells by the PEX tool. Subsequently, a dedicated set of PEX tool settings are derived to identify the defect locations. In Section 2.5.2, we present the algorithms to determine the full set of defect locations for both interconnects and transistors. The defect collapsing will be discussed in Chapter 3.

2.5.1 Parasitic Extraction For Defect-Location Identification

To illustrate the PEX methodology and utilize it to assist the DLI, we define a model for library cells as follows.

A *net* that electrically connects two or more *terminals* is divided into net *segments*, which are between two terminals and/or *internal nodes*. Let *Cells* be the set of library cells. Let $Terminals_c$ be the set of terminals of library cell $c \in Cells$. $Terminals_c$ contains the inputs and outputs of cell c (a.k.a. *ports*), power (V_{DD}) and ground (V_{SS}), and the source, drain, gate, and bulk terminals of the transistors in c. Let $IntNodes_c$ be the set of all internal net nodes of library cell c. Internal nodes are the points in a net where the PEX tool starts a new net segment. With $Nodes_c$ we denote the set of all nodes, i.e., $Nodes_c = Terminals_c \cup IntNodes_c$.

Parasitic resistors and capacitors can be represented by two weighted undirected graphs, $G_R = (V, R)$ and $G_C = (V, C)$, with a common set of vertices $V = Nodes_c$, but different edge functions R, C: $Nodes_c \times Nodes_c \longrightarrow \mathbb{R}^+$ where $R(i, j)$ and $C(i, j)$ specify the parasitic resistance and capacitance between nodes i and j respectively as extracted by the PEX tool.

The function *segm*: $Nodes_c \times Nodes_c \longrightarrow Bool$ denotes that nodes i and j are electrically connected by a net segment, i.e. $R(i, j) > 0$. *segm* is commutative, i.e.

$$segm(i, j) = segm(j, i) \tag{2.1}$$

The function *conn_net*: $Nodes_c \times Nodes_c \longrightarrow Bool$ is the transitive closure of *segm*, and denotes that two nodes are electrically connected through zero or more net segments, and is defined as follows:

$$\begin{aligned} conn_net(i, j) \equiv (i = j) \ \vee \ segm(i, j) \vee \\ \exists_{k \in Nodes_c}(conn_net(i, k) \ \wedge \ conn_net(k, j)) \end{aligned} \tag{2.2}$$

The function *net*: $Nodes_c \longrightarrow \mathcal{P}(Nodes_c)$ defines, for a given node i, the *net* of i, i.e., the set of nodes with which i is electrically connected in cell c:

$$net(i) = \{j \in Nodes_c | conn_net(i, j)\} \tag{2.3}$$

The net function effectively partitions the set $Nodes_c$ in a number of disjoint nets, such that the following holds:

$$\begin{aligned} \forall_{i \in Nodes_c}(i \in net(i) \ \wedge \ |net(i)| \geq 2) \wedge \\ \forall_{i,j \in Nodes_c}(j \in net(i) \Rightarrow i \in net(j)) \end{aligned} \tag{2.4}$$

The set of disjoint nets in cell c is $Nets_c = \bigcup_{i \in Nodes_c} \{\{net(i)\}\}$. Each node $n \in Nodes_c$ resides in a processing layer. Let *Layers* be the set of processing layers. Layer function

lyr: $Nodes_c \longrightarrow Layers$ gives the layer of a node. For $i, j \in Nodes_c$ with $segm(i, j)$, either (1) $lyr(i) = lyr(j)$ and the net segment between i and j is said to be in $lyr(i)$, or (2) $lyr(i) \neq lyr(j)$ and the net segment between i and j is a vertical interconnection between different layers, called a *via* or *contact*. The PEX tool can list the x, y layout coordinates of each node.

A net can fork to multiple destinations. We define *Forks*, the set of *fork nodes* in cell c, as follows:

$$\begin{aligned} &Forks_c = \\ &\{i \in Terminals_c \mid |\{j \in net(i) | segm(i, j)\}| > 1\} \cup \\ &\{i \in IntNodes_c \mid |\{j \in net(i) | segm(i, j)\}| > 2\} \end{aligned} \tag{2.5}$$

Figure 2.8 presents a net with four terminals and two internal nodes to shows the two types of fork nodes. The terminals *1*, *3*, and *4* are connected to one segment only, while the terminal *2* connects two segments. Only the terminal *2* among the four terminals is a fork node. This type of net is used to connect parallel transistor gate, source, or drain terminals. For example, in Figure 2.8, the terminal *1* can be an input port, and the terminals *2*, *3*, and *4* can be gate terminals of three parallel PMOS transistors (the NMOS terminals are omitted here). An internal node connects at least two segments. If an internal node connects more than two segments, it is a fork node, like the internal node 2 in figure 2.8.

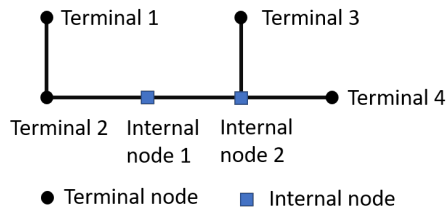


Figure 2.8: Fork terminal and internal nodes.

The function *conn_branch*: $Nodes_c \times Nodes_c \longrightarrow Bool$ denotes that two nodes are part of the same net branch, and is defined as follows:

$$\begin{aligned} &conn_branch(i, j) \equiv ((i = j) \vee segm(i, j) \vee \\ &\exists_{k \in Nodes_c \setminus Forks_c}(conn_branch(i, k) \wedge conn_branch(k, j))) \end{aligned} \tag{2.6}$$

The function *branch*: $Nodes_c \longrightarrow \mathcal{P}(Nodes_c)$ yields, for a given node i, *the branch of* i, i.e., the set of nodes that are part of the same net branch as i:

$$branch(i) = \{j \in Nodes_c | conn_branch(i,j)\} \tag{2.7}$$

The set of all branches in cell c is $Branches_c = \bigcup_{i \in Nodes_c} \{\{branch(i)\}\}$. A branch is bounded with two nodes that are terminal or fork nodes.

We illustrate the model described above with an inverter cell INVX1 from the Cadence GPDK045 cell library [Cadence Design Systems, 2014] as example in Figure 2.9. Figure 2.9(a) shows the layout. The twelve terminals are marked in the schematic (see Figure 2.9(c)): one input port (A), one output port (Y), two transistors with four terminals each, V_{DD}, and V_{SS}. Figure 2.9(b) shows in the PEX model each terminal exists in one of four layers: n-diffusion, p-diffusion, poly, and metal-1. V_{DD}/V_{SS} and cell input/output ports are on the metal-1 layer. In addition, the PEX tool has identified eight internal nodes. The total set of 20 nodes partitions into four disjoint nets: $Nets_{\text{INVX1}} = \{\{A, A\#1, A\#2, G1, G2\}, \{Y, Y\#1, Y\#2, D1, D2\}, \{V_{DD}, V_{DD}\#1, V_{DD}\#2, S1, B1\}, \{V_{SS}, V_{SS}\#1, V_{SS}\#2, S2, B2\}\}$. Contacts exist from metal-1 to the other three layers. Net $\{A, A\#1, A\#2, G1, G2\}$ forks out to two gate terminals $G1$ and $G2$, and contains three branches $\{\{A, A\#1, A\#2\}, \{A\#2, G1\}, \{A\#2, G2\}\}$. Similarly, the other nets also can be broken up into branches.

In cells more complex than INVX1, two adjacent transistors in the layout are designed to share the same diffusion area as their source or drain. However, in the DSPF format

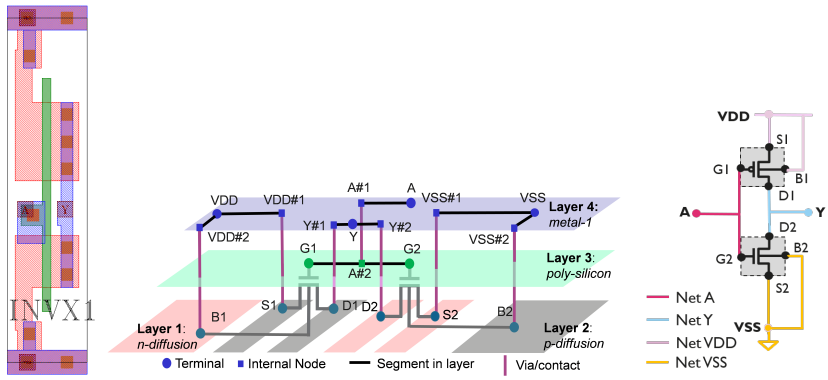


Figure 2.9: Example library cell INVX1: (a) layout, (b) schematic, and (c) PEX cell model.

netlist, the PEX tool assigns a particular name for each transistor terminal even though two different terminal names represent the same diffusion area. Consequently, the interconnection between such two terminals is modelled as a *virtual* net of 0.001Ω. The 0.001Ω resistor does not physically exist, and the resistance 0.001Ω is too small to impact the simulation results. Figure 2.10 shows an example virtual interconnect in the AND2X1 cell, designed as a NAND circuit followed by an inverter, from Cadence's GPDK045 library [Cadence Design Systems, 2014]. The two NMOS transistors in the NAND part share the same diffusion area, but without a physical interconnect between them. The interconnection between the terminals *S* and *D* shown in the schematic is extracted as a virtual net.

We use Cadence Quantus v18.1 to perform PEX. Figure 2.11 lists the general PEX settings. The remaining PEX settings specific to short- and open-defect location identification are described in the following two sections. Users should set the input database format generated by a layout versus schematic EDA tool. We use the '*pvs*' database generated by Cadence Pegasus (Line 1). We need to extract resistor (R) and capacitor (C) values (Line 2). We use DSPF (Cadence's detailed standard parasitic format) as output format (Line 3). The benefits in the CAT context of DSPF over, for example, the well-known Spice format [Stanford EE Handouts, 2001] are that the DSPF format [Cadence Design Systems, 1999] contours (1) an explicit layer list, (2) explicit listing of nets, and (3) explicit listing of net segments in diffusion layers. For usage during diagnosis, we include as comments the netlist for the parasitic Rs and Cs: layer information, segment widths, and x, y location coordinates (Lines 4–6).

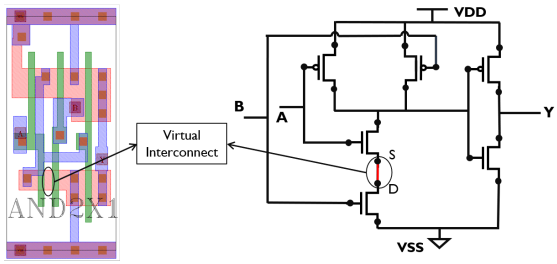


Figure 2.10: An example virtual net in the GPDK045 library cell AND2X1.

Quantus Settings 1 [GENERAL PEX SETTINGS]

```
1: input_db -type pvs;
2: extract -type rc_coupled;
3: output_db -type dspf;
4: output_db -include_parasitic_res_model comment;
5: output_db -include_parasitic_res_width true;
6: output_db -output_xy PARASITIC_RES PARASITIC_CAP;
```

Figure 2.11: General Cadence's Quantus settings for cell-aware DLI.

An open defect on an interconnect might occur between any two connected nodes (i, j) with *segm*(i, j). We analyze for each library cell its layout by letting the PEX tool extract all net segments and their parasitic resistor values. Each extracted segment *segm*(i, j) with its parasitic resistor $R(i, j)$ is considered as a potential open-defect location on the interconnects. There are three scenarios for which we need the PEX tool to split a net into multiple segments. (1) A *fork* in a net with multiple destinations. Open defects on different *branches* of the fork affect different terminals, and hence should be separated. (2) A vertical interconnect from one layer to another (referred to as a *via* which in library cells is typically called "contact"). (3) A 90° bend ('L shape') within a layer, as the contacts and the 90° bends are sensitive locations of manufacturing defects.

Figure 2.12 lists the settings of the user options of Quantus to identify the full set of open-defect locations on intra-cell interconnects. In Line 1, we instruct Quantus *not* to break up segments into multiple segments only because the layout length of the segment exceeds a limit. The two options in Lines 2–3 specify that defect-sensitive circuit elements are included in the PEX output netlist: contacts and 90° bends in nets – the latter is identified by electromagnetic analysis.

Parallel segments (other than parallel vias and contacts) are expected not to occur in the heavily-optimized circuits that standard-cell libraries are (Line 4). The only type of parallel R segments that occur frequently in library cells are parallel vias. The inherent redundancy of multiple parallel vias is meant to increase the yield, and effectively work as one (lower-resistance) parallel connection. Lines 5–6 merge them into one segment with replacement R value. To control complexity, our CAT flow works on the basis of one defect insertion at a time. However, by varying the resistance value of the open defect in the merged via, we can model any number of parallel vias as being defective. We suppress listing of parasitic resistors in dangling segments (Line 7). As the missing resistances do not affect the simulation results and open defects in dangling segments would not cause faulty behavior, we utilize this option. This option does not affect the

Quantus Settings 2 [FOR OPEN-DEFECT LOCATION IDENTIFICATION]

```
1: extraction_setup -max_fracture_length infinite;
2: output_db -add_explicit_vias true;
3: output_db -em_extract true;
4: filter_res -merge_parallel_res true;
5: filter_res -merge_parallel_via true;
6: filter_res -max_via_array_size auto;
7: filter_res -remove_dangling_res true;
8: filter_res -min_res 1e-10;
```

Figure 2.12: Dedicated Quantus settings for open-defect location identification.

extraction of capacitors to the dangling segment nodes, and therefore it does not affect the fault simulation results.

A R value is indicative of the probability that the corresponding segment will suffer from an open defect: a large R value indicates that the corresponding net segment is thin and/or long and both increase the probability for that interconnect to be affected by an open defect. The parameter `min_res` in Line 8 filters out parasitic resistors below a specified limit (in Ohm) during PEX. We want the PEX tool to include *all* resistor segments in the cell model as open-defect locations, and set a user-defined option R in the DLI function to filter out some open-defect locations with small Rs based on users' requirements. Quantus accepts a `min_res` value only larger than 0, therefore we specify `min_res` low enough (i.e. 10^{-10} Ω) to guarantee that all segment Rs are extracted.

To identify the short-defect locations between intra-cell interconnects, we let the PEX tool extract the parasitic coupling capacitors between all nodes in the library cell; Figure 2.13 shows the associated user options for Quantus. All self-coupling capacitors between nodes on the same net are extracted to obtain an accurate simulation model (Line 1). The DLI script will ignore these self-coupling capacitors for short-defect location identification, as we do not want to consider short defects between nodes on a single net because they are harmless. Floating nets are expected not to occur in the heavily optimized circuits like library cells. However, just in case they do occur any way, the command in Line 2 removes them as a single short connected to floating nets cannot cause a fault. We want a simple cut-off threshold to filter out capacitors for which the nodes which are too far apart in the cell's layout. The probability that these far apart nodes become shorted by a defect that would affect the interconnects is low. The Quantus `MinC` function does *not* provide such a straightforward cut-off filter. Hence, we set `MinC` as low as possible to extract all capacitors and apply our own cut-off threshold for

capacitor values in the Modus DLI script. Quantus does not accept `MinC`=0, and hence we set `MinC`=10^{-25} Farad below which Quantus does not extract any capacitance value (Lines 3–5).

Quantus Settings 3 [FOR SHORT-DEFECT LOCATION IDENTIFICATION]

```
01: filter_cap -exclude_self_cap False;
02: filter_cap -exclude_floating_nets true;
03: filter_coupling_cap;
04:  -coupling_cap_threshold_absolute 1e-25;
05:  -coupling_cap_threshold_relative 1e-3;
```

Figure 2.13: Dedicated Quantus settings for short-defect location identification.

2.5.2 Full-Set of Defect Location Identification

A standard cell is built up from interconnected transistors. We consider open and short defects for both interconnects and transistors. We first address the open-defect locations and then handle the short-defect locations.

With the dedicated user settings of the PEX tool, parasitic resistors indicate the potential open-defect locations on cell-internal interconnects. For transistors, we model transistor-internal defects as transistor-boundary defects. For example, bad doping causes high impedance at source or drain, which can be modeled by adding high-ohmic resistors at source or drain terminal [Khatri et al., 2016]; corrosion of the gate metallisation or poor gate etching results in a loss of the gate controllability, which can be modeled by gate terminal open [Galiay et al., 1980, Fantini and Morandi, 1985].

Algorithm 2.1 computes the full set of open-defect locations. This algorithm has several controls with which the user can authorize the tool to ignore certain open locations. The default settings are such that no locations are filtered out and thus best test quality can be obtained.

- Parameter *disableTrTerminalOpens* allows to explicitly disable the identification of open-defect locations on transistor terminals (default: *false*).
- *TrTerminalOpenSet* specifies which set of transistor terminals are considered as possible open locations (default: {*source, drain, gate*}).
- Threshold function $R_{\mathrm{th}}(\ell_1, \ell_2)$ defines, for each layer ℓ_1 if $\ell_1 = \ell_2$ or between two layers (ℓ_1, ℓ_2) if $\ell_1 \neq \ell_2$, the minimum value of segment resistors to be identified as open-defect locations (default: $R_{\mathrm{th}}(\ell_1, \ell_2) = 0$ for all layers ℓ).

In Line 2 of Algorithm 2.1, the full set of open-defect locations is initialized. Open-defect locations are identified on all physical segments of which the parasitic resistor values exceed the user-defined threshold $R_{\text{th}}(\ell_1, \ell_2)$ (Lines 03–07). Next, if the user has not disabled the identification of transistor open locations, for all transistors, the open-defect locations are identified on all terminals in *TrTerminalOpenSet* (Lines 08–12).

Algorithm 2.1: IDENTIFY THE FULL SET OF OPEN-DEFECT LOCATIONS

```
01: for all c ∈ Cells do {
02:   openFullSet_c = ∅;
03:   for all i, j ∈ Nodes_c do {
04:     if segm(i, j) ∧ segment (i, j) is not virtual then {
05:       if R(i, j) ≥ R_th(lyr(i), lyr(j)) then {
06:         openFullSet_c = openFullSet_c ∪ {(i, j)};
07:   } } }
08:   if ¬ disableTrTerminalOpens then {
09:     for all transistors t in c do {
10:       for all tm ∈ TrTerminalOpenSet do {
11:           openFullSet_c = openFullSet_c ∪ {t.tm};
12: } } } }
```

For intra-cell interconnects, we guide the PEX tool to extract all parasitic capacitors between net nodes and consider each parasitic capacitor $C(i, j)$ with *conn_net*(i, j) = *False* as a potential interconnect short defect location. We model typical transistor-inside defects, gate-oxide breakdown and improper diffusion, as short defects between transistor terminals. Kaczer et al. demonstrated that the gate-oxide breakdown between gate and source or drain can be exactly modeled by a short resistor between gate and source or drain terminals [Kaczer et al., 2003]. Galiay et al. showed that improperly large diffusion area of source and drain due to bad doping may cause a short defect between source and drain [Galiay et al., 1980]. The parasitic capacitors between interconnects and the pairs of transistor terminals constitute the full set of short-defect locations per cell.

Our function of short-defect location identification has the following user controls with which the user can authorize the tool to filter out certain short locations.

- Parameter *disableTrTerminalShorts* allows to explicitly disable the identification of short-defect locations on transistor terminals (default: *false*).
- *TrTerminalShortSet* specifies which set of transistor terminal pairs are considered as possible short locations (default: {*gate-source, gate-drain, gate-bulk, source-drain*}).
- A *blocklist* specifies layer pairs between which shorts should *not* be considered.

This list is technology dependent. (default: empty list).
- Threshold value C_{th} defines that extracted parasitic capacitors will not be identified as short-defect locations if $C < C_{\mathrm{th}}$ (default: $C_{\mathrm{th}} = 0$).

Algorithm 2.2: IDENTIFY THE FULL SET OF SHORT-DEFECT LOCATIONS

```
01: for all c ∈ Cells do {
02:   shortFullSet_c = ∅;
03:   for all a, b ∈ Nets_c with a ≠ b do {
04:     for all i ∈ a and j ∈ b do {
05:         if ((lyr(i), lyr(j)) ∉ BlockList) ∧ (C(i,j) ≥ C_th) then {
06:           shortFullSet_c = shortFullSet_c ∪{(i,j)};
07:   } } }
08:   if ¬ disableTrTerminalShorts then {
09:     for all transistors t in c do {
10:       for all transistor terminal pairs (tm_1, tm_2) ∈ TrTerminalShortSet do {
11:         shortFullSet_c = shortFullSet_c ∪ {(t.tm_1, t.tm_2)};
12: } } } }
```

Algorithm 2.2 identifies the full set of short-defect locations within a library cell. In Line 02, for cell c, the full set of short-defect locations is initialized. Subsequently, the parasitic Cs are evaluated to identify locations for short defects between intra-cell nets (Lines 03–07). For each net pair $a, b \in \mathit{Nets}_c$, if the node pair of $i \in a$ and $j \in b$ is not on the blocklist and for which holds $C(i,j) \geq C_{\mathrm{th}}$, we add it to the full set (Lines 05–06). Next, for all transistors, short-defect locations between the terminals of *TrTerminalShortSet* are identified, only if not disabled by the user (Lines 08–12).

Figures 2.14 and 2.15 show the extracted netlist file and the defect-location file of the GPDK045 AND2X1 cell's generated by Cadence's CAT tool flow. In Figure 2.14, we show the extracted parasitic resistors and capacitors only for Net A of the AND2X1 cell. All the other nets have the same format parasitic list. The first line in Figure 2.14 gives the net name and the extracted capacitance between the net and substrate. The following lines starting with symbol $\star$ are the list of terminals and nodes in Net A. The lines starting with 'C' present the extracted capacitors. In each capacitor line, the coupling node pair and the capacitance value follow the capacitor number. At the end, the coordinates of the midpoint of the two coupled nodes are given. Similarly, the lines staring with 'R' provide information of the extracted resistors. Following the resistor number, two nodes of the segment, resistance value, name of the layer in which the segment is, the width and length values of the segment, and the XY coordinates of the midpoint of the segment are listed.

In Figure 2.15, the simulation language is specified first. 'bridge' instead of 'short' is the

```
*|NET A 3.02037e-16
*|P (A X 0 0.1000 0.8550) // $llx=0.1000 $lly=0.8550 $urx=0.1000 $ury=0.8550 $lvl=1
*|I (Mmn1#g Mmn1 g X 0 0.2525 0.5125) // $llx=0.2525 $lly=0.5125 $urx=0.2525 $ury=0.5125 $lvl=2
*|I (Mmp0#g Mmp0 g X 0 0.1825 1.2225) // $llx=0.1825 $lly=1.2225 $urx=0.1825 $ury=1.2225 $lvl=2
*|S (A#3 0.1825 0.7950) // $llx=0.1825 $lly=0.7950 $urx=0.1825 $ury=0.7950 $lvl=2
*|S (A#4 0.1150 0.7950) // $llx=0.1150 $lly=0.7950 $urx=0.1150 $ury=0.7950 $lvl=2
*|S (A#5 0.2525 0.7925) // $llx=0.2525 $lly=0.7925 $urx=0.2525 $ury=0.7925 $lvl=2
*|S (A#9 0.1150 0.7950) // $llx=0.1150 $lly=0.7950 $urx=0.1150 $ury=0.7950 $lvl=1
C222 Mmp0#g VSS! 4.60376e-20 $X=0.1915 $Y=0.984
C223 Mmn1#g VSS! 7.43181e-19 $X=0.2465 $Y=0.3445
C224 A#9 VSS! 7.16233e-19 $X=0.11 $Y=0.685
C225 A#3 VSS! 9.1498e-20 $X=0.188 $Y=0.7525
C226 A#5 VSS! 2.34072e-20 $X=0.2495 $Y=0.678
C227 A#4 VSS! 2.07264e-19 $X=0.1205 $Y=0.691
C228 A VSS! 6.34487e-19 $X=0.1175 $Y=0.757
C229 A#4 n0#16 2.98094e-19 $X=0.1255 $Y=0.709
C230 A#4 Mmp0#b 1.57448e-18 $X=0.109 $Y=0.8085
C231 A Mmp2#g 4.07384e-20 $X=0.293 $Y=1.004
C232 Mmp0#g Mmp0#s 1.92284e-17 $X=0.1585 $Y=1.2835
C233 A#5 Mmp0#s 6.58218e-20 $X=0.2285 $Y=0.86
C234 A#4 n0#14 2.50837e-19 $X=0.1395 $Y=0.886
C235 Mmp0#g Mmn0#g 1.21035e-20 $X=0.25 $Y=0.925
C236 A#4 Mmp0#d 1.57621e-19 $X=0.1495 $Y=0.9075
C237 A#5 VDD!#3 6.5046e-21 $X=0.1965 $Y=1.057
C238 Mmp0#g VSS!#3 1.54971e-20 $X=0.332 $Y=0.8225
C239 A#5 n0#16 1.25697e-19 $X=0.231 $Y=0.695
C240 Mmn1#g B#5 7.31278e-19 $X=0.3365 $Y=0.714
C241 A Mmp1#s 9.95165e-20 $X=0.189 $Y=0.9635
...
Rk7 A A#9 0.035464 $metal1_conn  $L=0.27 $W=0.1055 $X=0.0975 $Y=0.825 $lvl=1
R18 A#3 A#4 5.230148 $poly_conn  $L=0.0075 $W=0.114 $X=0.1485 $Y=0.795 $lvl=2
R19 A#3 A#5 9.390481 $poly_conn  $L=0.0475 $W=0.109 $X=0.2175 $Y=0.795 $lvl=2
R110 A#5 Mmn1#g 78.228020 $poly_conn  $L=0.15 $W=0.039 $X=0.2525 $Y=0.6525 $lvl=2
R111 Mmp0#g A#3 120.000000 $poly_conn  $L=0.32 $W=0.039 $X=0.1825 $Y=1.0085 $lvl=2
Rx12 A#4 A#9 45.000000 $cont_poly  $L=0.06 $W=0.06 $N=1 $A=0.0036 $X=0.115 $Y=0.795 $lvl=6
```

Figure 2.14: DSPF-format netlist file extracted for cell AND2X1 in the GPDK045 library.

keyword that is used by the Spectre simulator. The default short-defect resistance, 0.001 Ω, is shown before the short-defect location list. Afterward, each defect-location line specifies the two shorted nodes and the short-defect resistance. The two shorted nodes correspond to an extracted capacitor. Similarly, the default open-defect resistance is 1G Ω and each open-defect location corresponds to a segment with an extracted resistor. Bridges 245–250 and Opens 13–18 show example formats of the shorts between and opens on transistor terminals.

2.6 Defect Simulation

At each open- or short-defect location, we inject a resistor to model the defect. In the case of a low-ohmic short or a high-ohmic open, we classify the defect as *hard*. Variants of the same defect with higher (for shorts) or lower (for opens) resistance exhibit a less impactful fault behavior than hard defects and hence are called *weak* defects. In

```
simulator lang=spectre

bridgeFault faults resBridge=0.001 {
  bridge {
    ...
    bridgeFault_bridge_222 ( circuit.Mmp0#g circuit.VSS! ) r=resBridge  // Mmp0#g VSS!
    bridgeFault_bridge_223 ( circuit.Mmn1#g circuit.VSS! ) r=resBridge  // Mmn1#g VSS!
    bridgeFault_bridge_224 ( circuit.A#9 circuit.VSS! ) r=resBridge  // A#9 VSS!
    ...
    bridgeFault_bridge_245 ( circuit.Mmn1#s circuit.Mmn1#d ) r=resBridge  // short source-drain MMmn1
    bridgeFault_bridge_246 ( circuit.Mmn1#s circuit.Mmn1#g ) r=resBridge  // short source-gate MMmn1
    bridgeFault_bridge_247 ( circuit.Mmn1#d circuit.Mmn1#g ) r=resBridge  // short drain-gate MMmn1
    bridgeFault_bridge_248 ( circuit.Mmp0#s circuit.Mmp0#d ) r=resBridge  // short source-drain MMmp0
    bridgeFault_bridge_249 ( circuit.Mmp0#s circuit.Mmp0#g ) r=resBridge  // short source-gate MMmp0
    bridgeFault_bridge_250 ( circuit.Mmp0#d circuit.Mmp0#g ) r=resBridge  // short drain-gate MMmp0
    ...
  }
}

openFault faults resOpen=1.000000e+12 {
  open {
    ...
    openFault_open_7 ( circuit.A ) r=resOpen { circuit.Rk7 }  // A A#9
    openFault_open_8 ( circuit.A#3 ) r=resOpen { circuit.Rl8 }  // A#3 A#4
    openFault_open_9 ( circuit.A#3 ) r=resOpen { circuit.Rl9 }  // A#3 A#5
    ...
    openFault_open_13 ( circuit.Mmn1#d ) r=resOpen { circuit.MMmn1 }  // open drain MMmn1
    openFault_open_14 ( circuit.Mmn1#g ) r=resOpen { circuit.MMmn1 }  // open gate MMmn1
    openFault_open_15 ( circuit.Mmn1#s ) r=resOpen { circuit.MMmn1 }  // open source MMmn1
    openFault_open_16 ( circuit.Mmp0#d ) r=resOpen { circuit.MMmp0 }  // open drain MMmp0
    openFault_open_17 ( circuit.Mmp0#g ) r=resOpen { circuit.MMmp0 }  // open gate MMmp0
    openFault_open_18 ( circuit.Mmp0#s ) r=resOpen { circuit.MMmp0 }  // open source MMmp0
    ...
  }
}
```

Figure 2.15: Defect-location file format.

general, the weaker the defect, the more difficult it becomes to detect. Modus allows user-defined defect resistances. By default, hard defect resistances are used. All open and short defects are modeled by 1GΩ and 0.001Ω, respectively.

After specifying the defect resistance values, we perform the analog simulation for all defects D in every library cell c. If we consider a single defect at a time, we simulate $|D|$ defects; if we consider all combinations of defects, we simulate $2^{|D|}$-1 combinations of defects. The simulation count increases exponentially with $|D|$, which makes the simulation impractical. If defect $d \in D$ presents a fault effect detected by cell pattern p, then typically all defect combinations containing d also can be detected by p, unless some defect in a combination masks the fault effect. The effectiveness of single fault assumption was demonstrated in literature [Hughes and McCluskey, 1984, Hughes, 1988, Bushnell and Agrawal, 2000] which proved that it was rare to happen that a fault effect was masked by another fault. Therefore, we simulate a single defect at a time, which makes the defect simulation practical.

We target static and delay faults caused by either short or open defects. For the static faults, one-cycle cell patterns are applied one by one at cell inputs and we observe the output logic values which are interpreted by the output voltage values. A delay fault

does not cause an incorrect cell output, but causes a delay in an output value transition. Therefore, we should apply two-cycle cell patterns to cause transitions at the cell outputs.

The simulation environment is shown in Figure 2.16. Each I/O is connected to a buffer cell. The usage of the buffer cells driving cell inputs is different from the buffers driven by the cell outputs. Input logic 1 is a high voltage equal to power voltage V_{DD}, while logic 0 is ground voltage V_{SS} = 0. The simulator offers ideal voltage sources with 0Ω resistance. In an actual circuit, library cell inputs are driven by another library cell instead of driven by ideal voltage sources, and the driving cell contributes resistance to the input of the driven cell. To mimic an actual circuit environment in the simulation, we use buffer cells as bridges between the ideal voltage sources and the inputs of the cell under simulation. The output buffer cells are used to strengthen the output voltages, such that we can easily interpret them as logic values. Figure 2.16 shows three example output voltage-time curves caused by three defects. In the two-cycle pattern simulations, we measure the transition delay at the output of the driven buffer cell and then subtract the correct output delay to obtain the extra delay caused by the inserted defect. The correct output delay is an accumulative cell delay from the input buffer cell to which we apply a signal transition to the output buffer cell. All cell delays are extracted from the standard-cell library timing files.

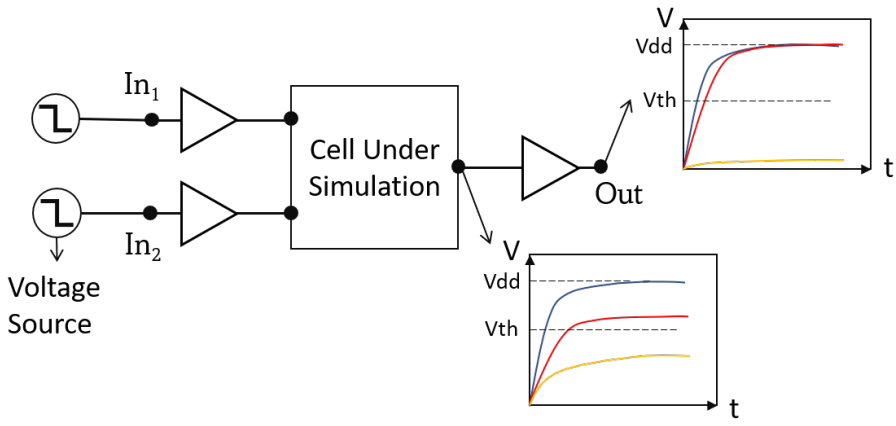


Figure 2.16: Simulation environment per library cell.

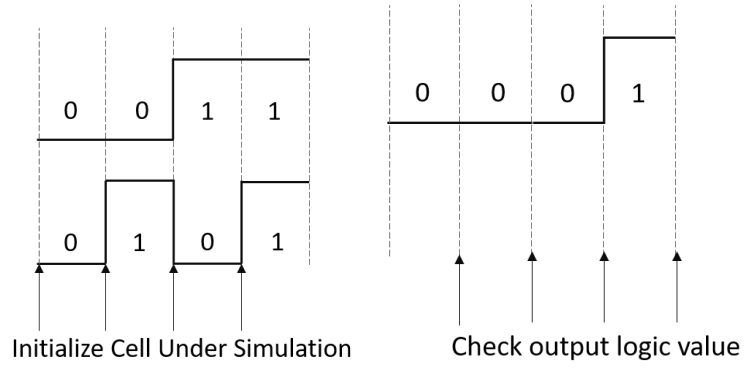
2.6.1 One-Cycle Pattern Detection

A cell with n inputs and m outputs has $m \times 2^n$ fully-specified one-cycle cell patterns to detect static faults observed at cell outputs. For each defect, we apply the 2^n stimulus patterns one by one and record the output logic values at m cell outputs. Figure 2.17 shows the total four stimulus patterns and their responses for an AND cell with two inputs and one output. We concatenate all the four stimulus patterns into one waveform. Transient fault analysis (TFA) function of Cadence's analog simulator Spectre® [Schaldenbrand et al., 2020] can initialize the cell under simulation when each cell pattern is applied, thus the circuit status caused by the previous cell pattern does not affect the simulation results for the subsequent cell pattern. Using one input pattern waveform per cell invokes the simulator only once for all defect-pattern tuples. Otherwise, the simulator needs to be invoked for each defect-pattern tuple, which is much slower.

Per defect-pattern tuple (d, p), if and only if the output logic value is opposite to the correct one at the output, we record DDM(d, p) = 1.

2.6.2 Two-Cycle Pattern Detection

In theory, a cell with n inputs has a total of 4^n - 2^n two-cycle stimulus patterns where at least one input has a transition. To observe the delay fault, we require the stimulus patterns to cause a transition for at least one cell output. Let a and b, with $a + b = 2^n$,



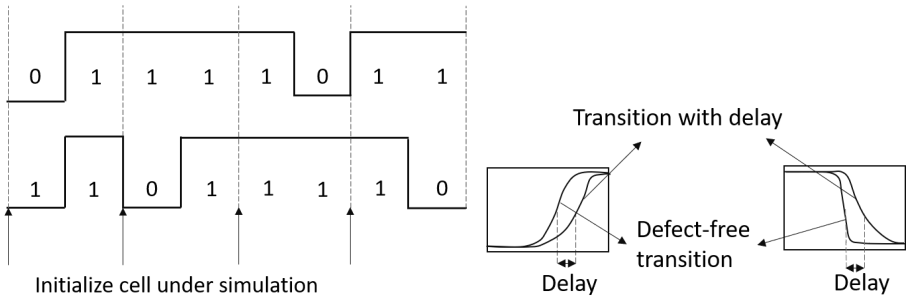
(a) One-cycle stimulus patterns.

(b) Correct output logic signals.

Figure 2.17: One-cycle detection of an AND cell with two inputs.

be the numbers of one-cycle stimulus patterns causing a cell output to be logic 0 and 1, respectively. The total number of two-cycle stimulus patterns with an output transition is $2 \times a \times b$. For example, cell AOI222 with 6 inputs has 27 and 37 one-cycle stimulus patterns leading to its output to be logic 0 and 1, respectively. Therefore, $2 \times 27 \times 37$ = 1,998 two-cycle cell patterns cause a rise or fall transition at the output. Simulating all the 1,998 two-cycle cell patterns for every cell-internal defect takes too long. We trim the set of $2 \times a \times b$ two-cycle cell patterns to a subset including the patterns with only one input change. The reason is as follows. Two-cycle patterns with multiple-input changes require the multiple-input transitions to arrive at the cell inputs simultaneously from chip inputs during testing. This requirement adds extremely strict timing constraints for the ATPG justification work, and therefore the expansions of two-cycle patterns with multiple-input changes are rarely achieved by the ATPG engine. Even though we would spend a lot of simulation time for these patterns, it is unlikely that they can be expanded successfully during ATPG. Hence there is no gain in test quality. We exclude the two-cycle patterns with multiple-input changes for the defect simulation.

We concatenate all two-cycle stimulus patterns into one waveform resetting the cell status with TFA every time. Figure 2.18(a) shows the input waveform for an AND cell with two inputs. The input two-cycle stimulus patterns are {R1, 1R, F1, 1F}, where 'R' and 'F' denote a rising and falling transition in the two clock cycles, respectively. At the cell output, we measure the additional delay compared to the delay of the defect-free cell, shown in Figure 2.18(b). Per defect-pattern tuple (d, p), if and only if the measured delay size is larger than the user-defined delay tolerance, we record DDM(d, p) = 1.



(a) Two-cycle stimulus patterns.

(b) Delay measurement in the second cycle.

Figure 2.18: Two-cycle detection of an AND cell with two inputs.

2.7 Cell-Aware ATPG

CA ATPG uses as inputs a cell-level netlist of a chip design and the set of DDMs of the cells occurring in that netlist (see Figure 2.3). The ATPG engine determines whether cell-level test patterns for the faults in DDMs can be expanded to chip level. Let I be the set of cell instances that occur in the chip's netlist and let function $t : I \longrightarrow C$ define the cell *type* of a given cell instance. For example, cell instance 'AND2X1_6' is the sixth instantiated AND2X1 cell. Therefore, $t(\text{AND2X1_6}) = \text{AND2X1}$. The set of faults F is defined by $F = \{(i, d) \in I \times D_{t(i)} | ps_{t(i)}(d) \neq \emptyset\}$.

The goal of CA ATPG is to expand at least one cell pattern $p \in \{ps_{t(i)}(d)\}$ for every fault $f = (i, d)$. The expansion consists of propagating the responses at the outputs of cell instance i to chip outputs, and justifying the specified stimuli at the inputs of i to chip inputs. Expansion might propagate and/or justify via other cell instances and their interconnecting nets. Successful expansion of a cell pattern for one fault in one particular cell instance typically covers other faults in the expansion path as well. Fault simulation can identify these cases and add the detections to the set of overall detections. If an expansion is not successful, typically there are alternative cell patterns for the fault in question, and their expansion can be tried as well. Only if expansion is unsuccessful for *all* cell patterns that detect fault f at the cell level, we declare fault f *undetectable*.

Conventional test methods, such as stuck-at (SA) and transition-delay (TD) tests, generate a test pattern starting from a fault location which is a cell I/O. The ATPG starts from a single propagation task and a single justification task. In CAT, the fault requires a cell pattern to activate it such that the fault is presented at a cell output. Therefore, the CA ATPG process starts from a cell pattern detecting the fault. In other words, it starts with a single propagation task at one cell output and n justification tasks at n cell inputs. More expansion tasks introduce more constraints and have a lower probability of successful expansion.

We use an example in Figure 2.19 to show the difference between SA ATPG and CA ATPG. SA test starts from sensitizing the fault at a cell input or output. If the SA fault is at a cell input, typically we need to assign some logic values to other cell inputs to propagate the fault effect. For example, in Figure 2.19(a), to propagate the SA-0 fault at input A of a multiplexer 2-to-1 cell, logic 0 should be assigned for input S, and input B is a don't-care bit. If the SA fault is at a cell output, we should assign inputs to justify the fault effect. In summary, the logic values at a cell inputs and outputs to detect a SA fault all depend on the cell's logic function and are independent of the technology. Instead, the CA fault detection at cell level is dependent on the technology. The CA defects are identified based on the cell layout which is specific to the technology, shown in Figure 2.19(b). The CA faults are activated and observed by performing analog

simulation which is based on the cell's electrical parameters, such as parasitic resistors and capacitors in cell-internal interconnects and transistors. These electrical parameters are also dependent of the technology. CA ATPG starts from propagating the faulty logic value at the cell output, and then justifies the logic values at cell inputs, according to the DDMs which are the outputs of analog simulation. For example, CA ATPG targets the short defect in Figure 2.19(b) and starts expanding the logic values of its only cell pattern, ABS/Y=101/L, identified by this short defect characterization.

Compared to SA and TD tests, CAT's effectiveness stems from the fact that CAT explicitly models all possible open and short defects based on the standard cells' layouts. The faults caused by these cell-internal defects normally include traditional SA and TD faults which are at cell boundaries. For example, in the CA static fault set the short defects between cell I/O and V_{DD}/V_{SS} cause faults equivalent to SA faults, and in the CA delay fault set the open defects at cell I/O cause faults equivalent to TD faults. The relationship between the sets of SA and CA static faults of a circuit is shown in Figure 2.20(a). In both SA and CA ATPG processes, a test pattern generated for a target fault typically can detect other faults. Please see more details in Section 2.1. Even though the SA

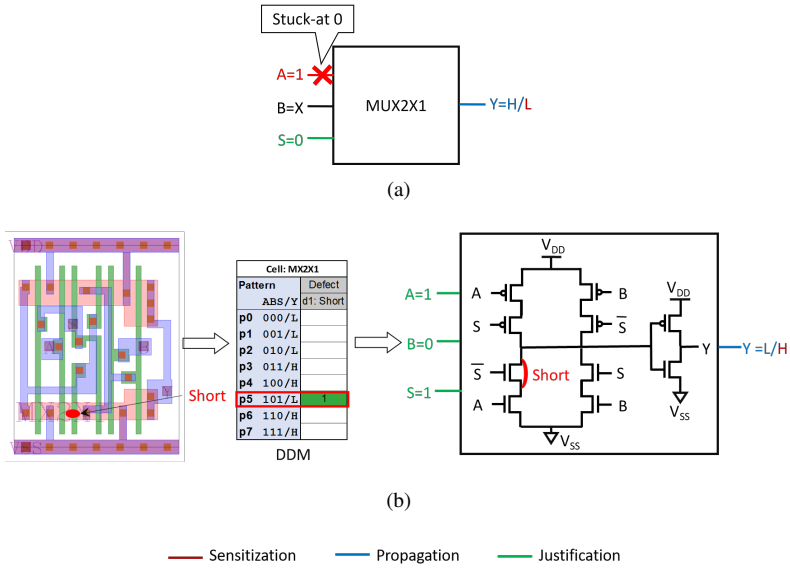


Figure 2.19: Difference between stuck-at ATPG and cell-aware ATPG. (a) Detecting stuck-at 0 fault at input A of an multiplexer 2-to-1 cell. (b) Detecting the source-drain short defect between the NMOS transistor controlled by ¬S.

ATPG does not explicitly target CA static faults, the generated SA test patterns still detect some non-SA faults serendipitously. As the CA ATPG targets all possible CA static faults identified in the library characterization, the CA test patterns cover not only SA faults but also more non-SA faults than the SA test pattern, shown as Figure 2.20(a). In a circuit design, the non-covered static faults by CAT in Figure 2.20(b) are due to the signal conflict in the ATPG process. For example, the short defect in multiplexer 2-to-1 cell shown in Figure 2.19(b) is detected by cell pattern ABS/Y=101/L only. If we use the SA test, the generated cell patterns may not include this cell pattern. Table 2.2(a) lists a SA pattern per SA fault of the multiplexer 2-to-1 cell. To cover all SA faults for this cell, we identify a SA pattern set that does not include the pattern ABS/Y=101/L, shown in Table 2.2(b). Therefore SA test cannot detect the short defect in Figure 2.19(b) which is identified and detected by CAT. Similarly, the CA delay test patterns detect more delay faults than the TD test patterns.

Compared to gate-exhaustive (GE) test, CAT's efficiency comes from the dedicated analog simulations which determine specific cell patterns for each defect, and therefore the ATPG tool expands no more cell patterns for the circuit than necessary to achieve fault coverage targets. For example, our cell-aware defect simulations for a 45nm and a 3nm technology AND2X1 cell both show that three out of the total four cell patterns already cover all intra-cell defects, while GE test requires all of the four cell patterns.

To quantitatively show the CAT's effectiveness and efficiency, we perform and compare SA, GE, and CA test with Cadence's ATPG tool Modus (version 19.1) on twelve circuits [Santosh Malagi, 2019, Scott Davidson, 1999, OpenCores.org, 1999, Arm, 2007] that were mapped onto Cadence's GPDK045 45nm CMOS library [Cadence Design

Figure 2.20: Relationship between SA test and CAT. (a) The set of CA faults is a superset of the SA faults. (b) ⓪ and ① constitute the full set of SA faults. ⓪ is the set of SA faults covered by the SA test patterns. ① is the set of SA faults non-covered by the SA test patterns and the CA test patterns. ② is the set of CA faults covered by the SA test patterns. ③ is the set of CA faults covered by the CA test patterns, which is a superset of ②. ④ is the CA faults non-covered by the CA test patterns.

Cell: MUX2X1	
SA Fault	**SA Pattern** I/O:ABS/Y
A SA-0	1X0/H
A SA-1	0X0/L
B SA-0	X11/H
B SA-1	X01/L
S SA-0	011/H
S SA-1	100/H
Y SA-0	1X0/H
Y SA-1	0X0/L

(a)

Cell: MUX2X1	
SA Pattern I/O:ABS/Y	**Covered SA Faults**
100/H	A SA-0, Y SA-0, S SA-1
010/L	A SA-1, Y SA-1
011/H	B SA-0, S SA-0
X01/L	B SA-1

(b)

Table 2.2: Stuck-at patterns for a multiplexer 2-to-1 cell. (a) SA pattern per SA fault. (b) The minimized set of SA patterns to cover all SA faults.

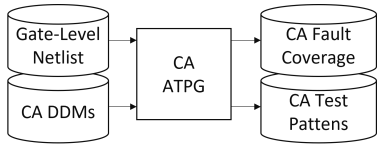
Patterns	**Defects**			
I/O:AB/Y	d1	d2	d3	d4
p0 :00/L	1	0	0	0
p1 :01/L	0	1	0	0
p2 :10/L	0	0	1	0
p3 :11/H	0	0	0	1

Table 2.3: Diagonal DDM for an AND2 cell.

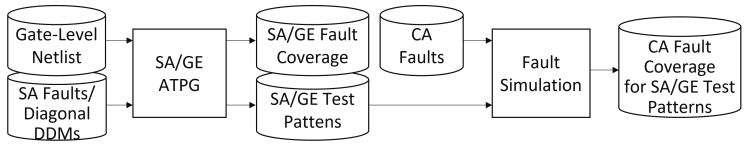
Systems, 2014].

Commercial ATPG tools do not support GE test, since it is impractical for industrial large-scale designs. To implement GE test, we flexibly utilize the CA ATPG engine in Cadence's ATPG tool Modus. During the CA ATPG process, the ATPG tool for each cell instance in a circuit tries to expand the cell patterns from cell to chip level to cover as many as possible cell-internal faults according to the DDMs. If all cell patterns need to be expanded successfully to cover all faults in a DDM of a cell, it is equivalent to performing GE test for this cell. Therefore, for each library cell, we build a dummy diagonal DDM in which the number of faults is equal to the number of cell patterns, and every fault is detected by an unique cell pattern. An example diagonal DDM of an AND2 cell is shown in Table 2.3. With the dummy diagonal DDMs as inputs and the goal of covering all faults in the DDMs, the ATPG tool will expand all cell patterns for each cell instance in a circuit. Thus, we implement GE test. As each fault is detected by unique cell pattern, the number of covered faults reported by the GE ATPG process equals to the number of successfully expanded cell patterns.

The SA fault coverage for a circuit is the ratio of covered SA faults to total SA faults, while the CA fault coverage is the ratio of covered CA faults to total CA faults. Comparing these two fault coverages is not fair for a test quality evaluation, as the total fault counts considered by SA ATPG and CA ATPG are different. To fairly compare the test quality between CA, SA, and GE, we use the CA fault coverage achieved by the CA, SA, and GE test patterns. Figure 2.21 shows how to obtain the CA fault coverage through the CA, SA, and GE ATPG flows. In the CA ATPG, this value is reported by the ATPG tool when the ATPG process is finished, shown in Figure 2.21(b). In the SA/GE ATPG flow, to get the CA fault coverage achieved by the SA/GE test patterns, we perform the fault simulation using the generated SA/GE test patterns against to the CA faults per circuit, shown in Figure 2.21(a).



(a) CA ATPG flow.



(b) SA or GE ATPG flow to obtain the CA fault coverage.

Figure 2.21: CA fault coverage based on SA, GE, and CA ATPGs.

Table 2.4 in which the circuits are sorted on the number of cell instances shows the various fault coverages. The average SA fault coverage is 97.17%, but the average CA fault coverage for the SA test patterns is only 95.85%. CAT considers more faults than SA test. Figure 2.22(a) shows the number of faults considered by SA test and CA test. On average, the number of CA faults is $\sim$6.0$\times$ SA faults. The average CA fault coverage for the SA patterns is 95.85% compared to 97.88% achieved by CA ATPG, namely the SA test misses 2.03% CA faults. In the other words, the CA test reduces the fault coverage escape from 1-95.85%=4.15% to 1-97.88%=2.12%. This constitutes a 48.9% reduction of fault coverage escape, which indicates a significant reduction of test escapes which is quantified in defective parts per million (DPPM). This result demonstrates Figure 2.20(b). The GE fault coverage on average is only 70.72%, which is the percentage of successfully expanded cell patterns in each circuit. [Hapke et al., 2011b] directly compares the GE fault coverage with the CA fault coverage to show the test quality, which is not meaningful as not faults but cell-pattern expansions are the targets of the GE test.

Design Data		Fault Coverage(FC)				
Name	#Instances	SA FC	CA FC for SA patterns	CA FC for CA patterns	GE FC	CA FC for GE patterns
b20	3,656	97.90%	94.04%	97.53%	67.03%	97.63%
b15	3,380	96.19%	95.16%	97.70%	77.42%	97.70%
aes	3,926	97.38%	98.69%	99.26%	67.78%	99.27%
b21	4,233	97.55%	92.56%	96.66%	77.61%	96.66%
dtmf	4,286	97.49%	97.57%	98.74%	71.24%	98.74%
b22	5,669	97.86%	93.67%	97.36%	77.92%	97.36%
M0	6,160	97.98%	98.43%	99.10%	71.72%	99.12%
b17	10,390	96.69%	95.63%	97.71%	66.95%	97.72%
b18	23,500	97.31%	97.38%	97.69%	67.89%	97.69%
fpu	24,141	97.05%	95.25%	97.50%	66.97%	97.54%
M3	38,616	95.57%	96.61%	97.71%	68.46%	97.80%
b19	46,210	97.07%	95.21%	97.61%	67.61%	97.64%
Avg.	14,514	97.17%	95.85%	97.88%	70.72%	97.91%

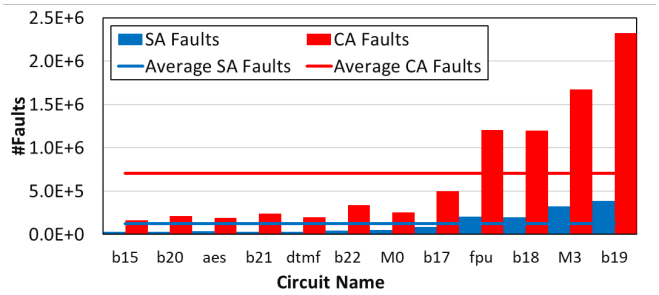
Table 2.4: The design data and fault coverages for twelve circuits.

With the fault simulation, we identify that GE test patterns achieve 0.03% higher CA fault coverage than the CA test patterns. As achieving the highest fault coverage with the minimized number of generated test patterns is an $\mathcal{NP}$-hard problem, the fault coverage and the set of test patterns generated and optimized by the APTG engine can vary in different runs. We can ignore the 0.03% fault coverage difference that may be caused by a reasonable variation of the ATPG results. We show only the CA fault coverages by SA, CA, and GE tests in Figure 2.22(b). In summary, CAT detects 2.03% more CA faults than SA test, and GE test detects the same number of CA faults compared to CAT.

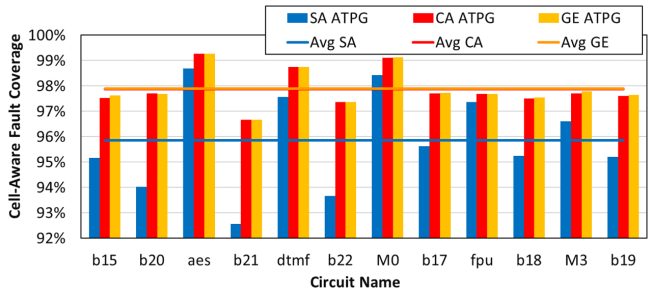
With a better test quality than SA test, CAT requires 28.1% more test patterns, shown in Figure 2.22(c). We also perform a top-off CA ATPG which targets the missed 4.15% CA faults to generate test patterns in addition to the SA test patterns. 48.6% more test patterns are generated by the top-off CA ATPG to achieve a CA fault coverage which is the same with that of the scratch CA test. GE test on average requires $\sim 2\times$ test patterns of CAT per circuit, shown in Figure 2.22(c). The test pattern count indicates the test execution time spent on an automatic test equipment (ATE) for every under-test-chip, hence under the condition that we achieve the same CA fault coverage, CA test significantly reduces the test cost compared to GE test.

The average ATPG compute time of SA, CA, and GE tests are 295.0 (= $1\times$), 547.5 (= $1.8\times$), and 728.7 (= $2.5\times$) seconds, respectively. The ATPG is performed once per circuit, and hence is an NRE. With the 48.9% reduction of fault coverage escapes, we consider the 80% ATPG compute time increase to be reasonable compared to SA test. Since GE ATPG expands more than necessary cell patterns to cover the CA faults, GE test takes longer compute time than the CA ATPG.

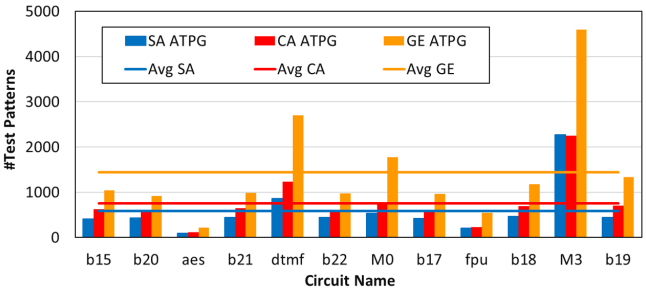
In summary, CA test achieves higher test quality with a slight increase in test cost com-



(a) Fault count.



(b) Cell-aware fault coverage.



(c) Chip-level pattern count.

Figure 2.22: Stuck-at test vs. cell-aware test vs. gate-exhaustive test.

pared to SA test, and achieves the same test quality with a much lower test cost compared with GE test.

2.8 Conclusion

In this chapter, we introduce the basic ATPG algorithm and the Cadence CAT tool flow that is used as the baseline of the optimization work in this thesis. Comparing with the traditional ATPG-based approaches, stuck-at test and transition-delay test that are the most popular methods used in the semiconductor industry, we theoretically analyze the reasons of CAT's effectiveness. The high test quality of CAT stems from explicitly targeting the faults caused by realistic defects per cell. The set of cell-aware static faults is normally a superset of the traditional stuck-at fault set, and the set of cell-aware delay faults is a superset of the transition-delay fault set. The intra-cell defects and their occurrence probabilities are identified based on cell layout. To avoid an unnecessarily large set of defects, we can exclude some locations with low defect probability which is reflected by the extracted parasitic values. To achieve the best test quality, we should consider all possible defect locations. Based on twelve benchmark circuits mapped on Cadence's GPDK045 45nm library, the ATPG results show that CAT reduces the non-covered faults by 48.9% at a cost of 28.1% more test patterns compared to the SA ATPG, and that CAT achieves the same fault coverage with half the number of test patterns compared to the gate-exhaustive ATPG. Therefore, we conclude that CAT reduces significant test escapes at an affordable cost.

CHAPTER 3

Defect Collapsing to Reduce Library Characterization Time

The full set of defect locations is large, even for small library cells. This makes the time cost of the defect simulation unaffordable. However, many defects in the full set are equivalent with respect to their logic fault behaviors, and hence require the same test patterns, causing identical DDM columns. For all equivalent defects, we need to simulate only one of them. Thus, the defect simulation time can be significantly reduced while the test quality is maintained. The process of selecting only one defect per equivalent defect group is called *defect collapsing* which generates a *compact set* from the full set of defect locations for the downstream defect simulation. The question is how to accurately identify equivalent defects.

In this chapter, we first propose a defect equivalence analysis. Subsequently, we use it to propose an approach consisting of defect collapsing and three-round simulations to minimize the number of simulated defects while maintaining test quality. The defect equivalence analysis shows that some collapsed defect groups may contain some non-equivalent defects. This leads to missing defects. To maintain test quality, we need to simulate all non-equivalent defects, such that all unique faults caused by the defects are

included into the DDMs. In the first round of simulation, we simulate the collapsed defects, and also use the simulation results to identify the defect groups that may be over-collapsed. This identification result contains some false positives. Subsequently, the identified defect groups undergo the second round of simulation to remove the false positives and obtain the real over-collapsed defect sets. In the third simulation round, we simulate the corresponding full set of defects per over-collapsed defect set to guarantee that we do not miss any defects.

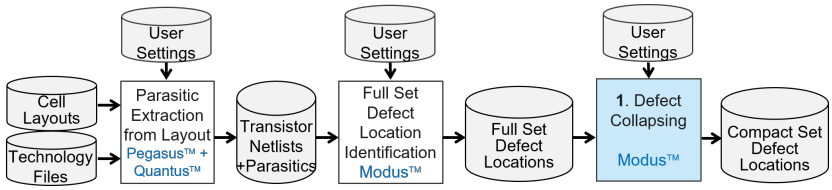
The remainder of this chapter is organized as follows. The proposed library characterization flow is introduced in Section 3.1. To reduce the defect simulation time and maintain the test quality, we describe how we handle the open and short defects in Section 3.2 and 3.3, respectively. Section 3.4 presents experiment results and Section 3.5 concludes this paper.

3.1 Proposed Library Characterization Flow

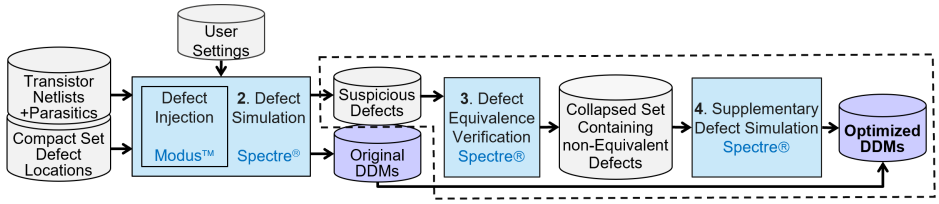
The proposed CAT library characterization flow is depicted in Figure 3.1. It consists of two steps: defect-location identification (DLI) and defect characterization. In the DLI flow shown in Figure 3.1(a), we introduce the parasitic extraction and the full set of DLI steps described in Chapter 2. In this chapter, we present the following steps in the library characterization. Compared to the baseline CAT tool flow introduced in Chapter 2, we add two more steps for the defect characterization, defect equivalence verification and supplementary defect simulation to optimize test quality, shown in Figure 3.1(b).

To save downstream defect characterization time while maintaining the test quality based on the full set of defects, we propose an approach consisting of four steps: (1) defect collapsing, (2) defect simulation, (3) defect equivalence verification, and (4) supplementary defect simulation, shown in Figure 3.1. The fundamental of this solution is based on *defect equivalence*. Equivalent defects are detected by the same set of cell patterns, and therefore can be collapsed into one representative defect that is to be simulated in the defect characterization. Step 1 is to collapse each group of equivalent defects, such the number of defects to be simulated is reduced and hence the defect characterization time decreases. Even though this step can drastically reduce the number of defects for the downstream defect simulation, some groups of defects may be over-collapsed because our defect equivalence identification is not as accurate as simulation. In Step 2, by the defect simulation, we identify the set of cell patterns for each defect representing a group of equivalent defects. Meantime, we use the simulation results to identify the suspicious defects corresponding to the groups of defects that are possibly over-collapsed. Steps 3 and 4 are two rounds of simulations verifying the defect collapsing result from Step 1 and

adding the missing DDM columns due to the over-collapsed defect groups, respectively, to maintain the test quality.



(a) Defect-Location Identification.



(b) Defect Characterization.

Figure 3.1: Library characterization flow. Steps 1 and 2 are the contributions to the baseline CAT tool flow in Chapter 2. Steps 3 and 4 inside the dotted line are additional optimization work.

3.2 Open-Defect Identification and Simulation

This section describes how we handle the full set of open defects to reduce the simulation time. Section 3.2.1 analyzes the electrical behaviors of open defects to identify their equivalence, which is the basis of open-defect collapsing and defect equivalence verification. The algorithm for open-defect collapsing is also described. Section 3.2.2 introduces the three-round simulations for open defects.

3.2.1 Open-Defect Equivalence and Collapsing

Open defects on different *branches* of a fork structure affect different downstream destinations, and therefore should be characterized separately. All open defects on a branch can be equivalent because they affect the same endpoints. We can therefore collapse all

these open defects into one. However, due to the impact of parasitic resistors and capacitors, the open defects on a branch can lead to slightly different electrical effects, i.e., different delays. If the delay variation causes not all opens on a branch to be detected by the same set of cell patterns, to maintain the test quality we should not collapse the opens on this branch. Since it is difficult to identify the branches with non-equivalent open defects without analog simulation and it is impractical to simulate the full set of open defects, we propose to collapse open defects into one for each branch, shown as Step 1 of the proposed flow in Figure 3.1, being aware of the fact that some branches with over-collapsed sets of open defects may exist. During Step 2, we record not only DDMs but also the delay caused by the simulated open defect per branch. These delay values are used to identify the suspicious defects that represent the branches on which the open defects may be over-collapsed. In Step 3, we use analog simulation to determine the branches with non-equivalent open defects among the suspicious branches, and simulate the full set of open defects for these branches in Step 4.

We analyze the electrical effect variation caused by different open defects on a branch in the following. Based on this analysis, we classify the branches into four types. Only one type of branch needs to go through Steps 3 and 4.

Type 1: A branch with only one segment. On such a branch, at most one open defect is determined, and we cannot collapse any open defects anyway. Therefore, the test quality is always maintained.

Type 2: A branch consists of multiple segments and the branch is a part of net V_{DD} or V_{SS}. On the power rails of standard cells in a circuit design, multiple voltage sources are designed and implemented with contacts between the external power supply and the power rails. An open defect cannot block the voltage source for transistors. The concept is shown in Figure 3.2. On the branch consisting of segments (0, 1) and (1, 2), an open defect can occur on either segment. Either R_{open1} on segment (0, 1) or R_{open2} on segment (1, 2) blocks the voltage source on the left side, but the voltage source on the right side still supplies power. Therefore, the two open defects on the branch that consists of segments (0, 1) and (1, 2) have the same effect on the circuit, and can be collapsed.

Type 3: A branch consists of multiple segments and an open defect on such a branch blocks the signal transitions for *all* downstream PMOS or NMOS gate terminals. A net can drive multiple parallel CMOS transistor gate terminals. An example net structure driving multiple transistors is shown in Figure 3.3. The net connecting six transistor drain terminals and six transistor gate terminals is split into 19 branches of which numbers are marked beside the branches. We use different colors for adjacent branches. Among branches 7, 10, 11, and 12 which consist of multiple segments, only branch 10 is of Type 3.

Type 4: A branch consists of multiple segments but open defects on this branch do *not* affect *all* downstream transistor gate terminals. This branch type can be found in standard cells with high drive strength, as multiple parallel transistors controlled by the same net are used to increase the drive strength. These multiple parallel transistors are switched on simultaneously, such that a large current through the transistors quickly charges or discharges the downstream components. In Figure 3.3, of branches 7, 10, 11, and 12 that consist of multiple segments, only branches 11 and 12 are of Type 4.

Open defects at different locations result in different *RC* networks of a branch. Figure 3.4(a) shows an example *RC* network of a defect-free branch. The capacitive coupling between the branch and ground net V_{SS} is extracted by the PEX tool as capacitors that are distributed over the branch nodes. Figure 3.4(b) shows two open defects at different locations on the branch. Only one is active at a time. As the branch's parasitic

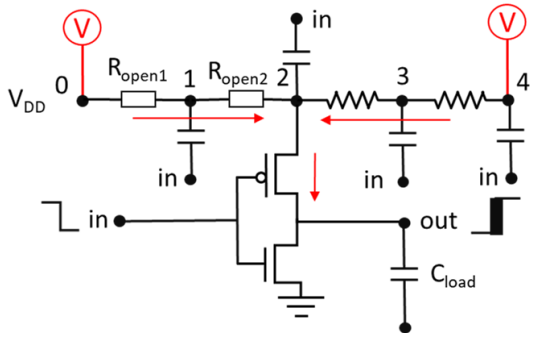


Figure 3.2: Different open locations on a power net branch.

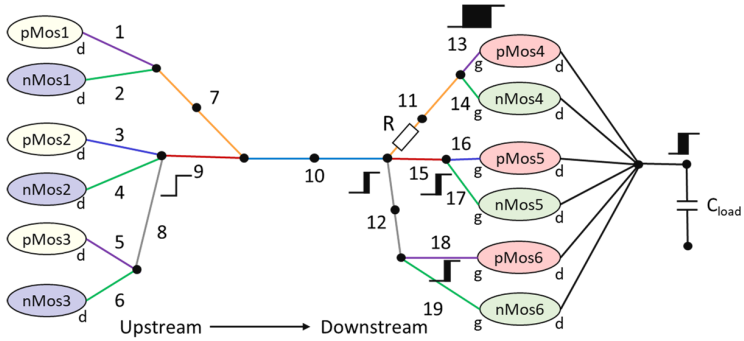


Figure 3.3: An example net in a cell with high drive strength.

resistance is significantly smaller than the open-defect resistance, the time constant of delay caused by either open defect is determined by the open-defect resistance and the capacitance from the open resistor to the downstream branch terminal [Rabaey et al., 2003]. If we assume node 4 is closer to the downstream transistor network than node 0, the time constants of the delays due to *open1* and *open4* are $R_{open1} \cdot (C_1 + C_2 + C_3 + C_4)$ and $R_{open4} \cdot (C_4)$, respectively. With $R_{open1} = R_{open4}$, the open-defect location affects the delay size through the variation of load capacitance calculated by summing the capacitance from the open-defect location up to the branch's downstream terminal.

Please note, in standard cells the load capacitance after an open defect is the sum of capacitances from the open-defect location up to the downstream transistors' gate terminals. It consists of two components: (1) interconnect capacitance and (2) the total gate capacitance of the transistors driven [Rabaey et al., 2003]. Figure 3.5 shows the load capacitances of net n2 that is driving an inverter. The two components of the load capacitance are drawn in red and blue colors. Eight nodes, internal nodes 0–5 and two gate terminals, split net n2 into seven segments and three branches. Net n2 has two neighboring nets, n1 and n3. The parasitic capacitances between n2 and the two neighboring nets are distributed based on each node of n2. Ci_j denotes the capacitance distributed between node i and neighboring net j. Ci_j is calculated by summing the values of extracted capacitors of which one coupling node is i and the other coupling node is on net j. Two open defects *open1* and *open2* are at different locations on the same branch of net n2 and only one open defect is present and simulated each time. Assuming the resistances of *open1* and *open2* are equal, *open1* causes a longer delay than *open2* due to the extra interconnect capacitors $C1_{n1}$ and $C1_{n3}$.

The interconnect capacitance after an open defect is also affected by the logic values on the neighboring nets. A resistive open defect causes a delay fault and hence can be detected by two-cycle test patterns. We perform simulations of a set of two-cycle cell patterns; each pattern contains only one cell input transition and causes a transition for

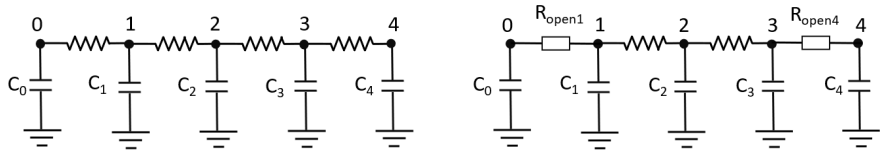


Figure 3.4: (a) *RC* network of a defect-free branch. (b) *RC* network with two opens at different locations.

at least one cell output. The two-cycle cell patterns cause either a static or a dynamic logic value on each neighboring net. In the following, we discuss the effect of static and dynamic neighboring nets on the interconnect capacitance.

Figure 3.6(a) shows a transition from 1 to 0 on net n2, while the logic value on its neighboring net n1 is static. The node capacitance between the open branch and its neighboring net is equal to the extracted capacitance values. Figure 3.6(b) illustrates opposing signal transitions on net n2 and its neighbor net n1. In the first cycle of the cell pattern, due to the voltage difference between nets n2 and n1, net n2 with its high voltage charges the parasitic capacitors between it and net n1. In the second cycle, the voltage on net n1 is high while the voltage on net n2 is low, therefore, the parasitic capacitor between n1 and n2 is first discharged and then charged. Therefore, the equivalent load capacitance between nets n1 and n2 are twice the extracted capacitance. Thus, neighboring nets with an opposite transition to the open branch increase the delay. Similarly, a neighboring net with the same transition with the open branch decreases the delay, shown in Figure 3.6(c). As the voltages on both n1 and n2 are the same in both clock cycles, the parasitic capacitor is neither charged nor discharged. The delays at the starting point and endpoint of net n2 are identical, and the equivalent capacitance is zero. [Arumí et al., 2008] define an indication function to evaluate the impact of logic state on neighboring nets. We adapt this function into our usage environment in which all nets are inside library cells and the logic values on the nets are determined by the cell pattern. Given a cell pattern p, our indication function $I_{x,l}(p)$ evaluates for a net x, the impact of logic value state on its neighboring net l, shown as Equation 3.1.

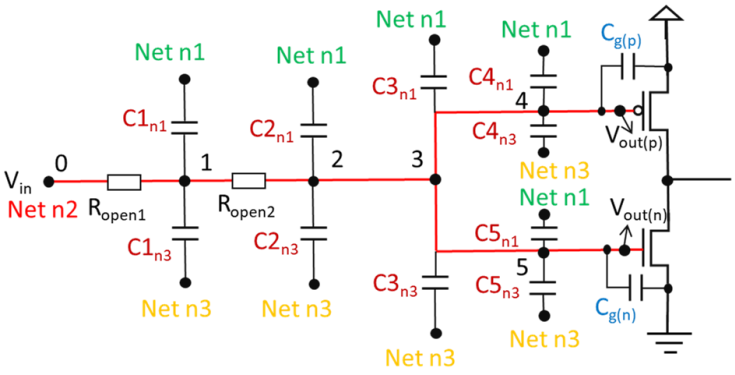


Figure 3.5: Circuit model of two opens at different locations on a branch.

$$I_{x,l}(p) = \begin{cases} 0, \text{when } x \text{ and } l \text{ have the same transition} \\ 1, \text{when } n \text{ does not transition} \\ 2, \text{when } x \text{ and } l \text{ have the opposite transitions} \end{cases} \tag{3.1}$$

Given an open defect on a net segment (i, j) where i and j are the two end nodes of the segment, the set of nodes from the open-defect location to all downstream gate terminals N can be obtained by tracking the nodes along the signal transmission direction. For net x, if one or more capacitors are extracted between x and another net l, we add l to the neighboring net set L of x. Therefore, the equivalent capacitance from open segment (i, j) to the end of all downstream gate terminals with respect to a cell pattern is calculated by

$$C_{(i,j)}(p) = \sum_{k \in N} \sum_{l \in L} (Ck_l \cdot I_{x,l}(p)) + \sum C_g \tag{3.2}$$

$\sum_{k \in N} \sum_{l \in L} (Ck_l \cdot I_{x,l}(p))$ is the interconnect capacitance after the open-defect location. The location of the open segment (i, j) determines the set of nodes after the open segment

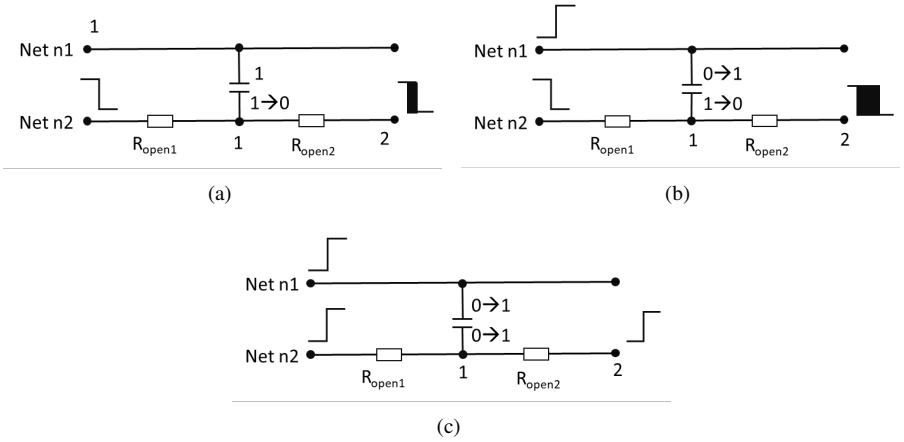


Figure 3.6: (a) Neighboring nets with a static logic value. (b) Neighboring nets with an opposite transition of the open net. (c) Neighboring nets with the same transition of the open net.

N, and cell pattern p determines $I_{x,l}(p)$. $\sum C_g$ is the sum of gate capacitances at the downstream terminals of the net.

On Type 1 and Type 4 branches, an open defect cannot block all signal propagation ways. The delay caused by an open defect on these branches is insignificant. Therefore, the delay variation on these branches can be ignored. The delay caused by an open defect on a Type 3 branch dominates the cell output delay, as the Type 3 branches are the only way for a signal transition to propagate to the cell output. For example, in Figure 3.3 the difference between open defects on branches 10 and 11 (or 12) is that an open defect on branch 10 affects all downstream transistors while an open defect on branch 11 only affects *pMos4* and *nMos4*. Open defects on the two segments of branch 10 cause different delays due to their different load capacitances. However, open defects on branch 11 do not block the signal for transistors *pMos5*, *nMos5*, *pMos6*, and *nMos6*, which still perform well. In a defect-free situation, either {*pMos4*, *pMos5*, *pMos6*} or {*nMos4*, *nMos5*, *nMos6*} are conducted simultaneously. With an open defect on branch 11, only {*pMos5*, *pMos6*} or {*pMos5*, *pMos6*} are conductive simultaneously. Even though an open defect results in a serious delay at the gate terminal of *pMos4* or *nMos4*, the delay at the output of the six transistors is determined by the performance difference between three and two conducting transistors. Before the signal is propagated to *pMos4* or *nMos4* gate terminal, the other transistors have already transferred the signal to the downstream circuit. Therefore, the delay variation caused by different open-defect locations on branch 11 has an insignificant effect on the cell output delay. Considering one open defect on branch 11 is sufficient. In summary, only for a Type 3 branch, the open on the most upstream segment causes the longest delay while the most downstream segment open causes the shortest delay.

In the defect characterization, we apply cell patterns sequentially for each open defect. For each cell pattern, only if the delay caused by the open defect is larger than the user-defined tolerated maximum delay, is the open defect detectable by the cell pattern. After we apply a cell pattern, if the delays caused by the most upstream and the most downstream segment opens on a branch are both larger (smaller) than the tolerated delay, all delays caused by the open defects on this branch are larger (smaller) than the tolerated delay. Hence, all open defects on the branch are detected (non-detected) by the applied cell pattern. If the open defects on the most upstream and the most downstream segments of a branch are equivalent, all opens on this branch are equivalent. Otherwise, not all open defects on this branch are equivalent, and we need to identify the multiple different DDM columns caused by these open defects.

With the assumption that all open defects per branch are equivalent, we first collapse the full set to a compact set of open defects by selecting the most upstream segment as the representative open defect per branch. For each cell pattern, based on the delay of the open at the most upstream segment given by the simulator, we derive the delay caused

by the most downstream open defect and then calculate the DDM column of the latter open. To verify if all opens on a branch equivalent, we compare the two DDM columns of the most upstream and the most downstream segment opens. If they are not identical, we run simulation for all segment opens on the branch to identify all different DDM columns.

The open-defect collapsing is described in Algorithm 3.1. To prevent modeling multiple open-defect locations on the same branch, Algorithm 3.1 stores which branches already have been assigned an open defect in array *branchOpens*. In Line 03, this array is initialized to *false*. In Lines 04–09, for each branch, the open location is identified on the most upstream segment. Line 05 adds the segments on which an open defect is identified into the full set. Two possible reasons causing no open defect identified on a segment are: (1) the segment is a virtual interconnect, (2) the extracted parasitic resistance of the segment is smaller than the user-defined resistance threshold, such that the segment is considered an impossible open-defect location. Next, for all transistors, open locations are identified for the compact set, if not disabled by the user (Line 09). In case a transistor terminal is a fork node, it becomes a transistor-open location that is non-equivalent to any branch-open defect (Line 12). For each of the non-fork transistor terminals, if no open-defect location is identified on the branch which includes the transistor terminal yet, we add an open-defect location on this terminal (Lines 13–15).

Algorithm 3.1: OPEN-DEFECT COLLAPSING

```
01: for all c ∈ Cells do {
02:   openCompSet_c = ∅;
03:   for all b ∈ Branches_c do { branchOpen[b] = False; }
04:   for all b ∈ Branches_c do {
05:     if ∃ segm(i, j) ∈ openFullSet_c, where i, j ∈ b then {
06:       identify the most upstream segment segm(i, j) ∈ openFullSet_c, where i, j ∈ b;
07:       openCompSet_c = openCompSet_c ∪ {(i, j)};
08:       branchOpen[b] = True;
09:   } }
10:   if ¬ disableTrTerminalOpens then {
11:     for all transistors t in c do {
12:       for all t.tm ∈ openFullSet_c do {
13:         if t.tm is a fork node then {openCompSet_c = openCompSet_c ∪ {t.tm};}
14:         else { if branchOpen[branch(t.tm)] = False then {
15:           openCompSet_c = openCompSet_c ∪ {t.tm};
16:           branchOpen[branch(t.tm)] = True;
17: } } } } } }
```

3.2.2 Open-Defect Simulation and Equivalence Verification

The first round simulation is performed only for the compact set of open defects. As discussed in Section 3.2.1, only for each branch in Type 3, we must verify that all open defects have identical DDM columns. During the first round simulation, per combination of Type 3 branch and cell pattern, we record the delay caused by the most upstream segment open and calculate the delay caused by the most downstream open defect based on the load capacitances ratio of the most upstream open to the most downstream open. If the delay caused by the most upstream segment open is larger than t_{th} while the delay caused by the most downstream segment open is smaller than t_{th}, the opens in the branch are not equivalent for this cell pattern, and hence cause different DDM columns. The equivalent capacitances for the two open defects are calculated by Equation 3.2. To simplify the calculation, we consider the largest capacitance difference between the two extreme open defects by using Equation 3.3 in which we always set the indication function value to be 2, which is the maximum value by $C_{x,l}(p)$ (see Equation 3.2). Equation 3.3 is thus a worst-case approximate of Equation 3.2, but much quicker to calculate without simulation. The calculated delay of the most downstream segment open defect is therefore often smaller or equal to the real delay, and therefore leads to a larger set of branches with non-equivalent segment open defects than the reality. If the calculated delay of the most downstream segment open defect is larger than t_{th}, the most upstream and the most downstream segment open defects are equivalent for sure. We call the branches with non-equivalent open defects identified by the delay calculation *suspicious* branches for which we run the second-round simulation to filter out the false positives. During the second-round simulation, we only simulate the most downstream segment open defect for each suspicious branch. With the correct delay given by the simulation, we identify the accurate set of branches with non-equivalent open defects, and send these branches to the third-round simulation that characterizes the full set of open defects for them.

$$C_{(i,j)} = \sum_{k \in N} \sum_{l \in L} (Ck_l \cdot 2) + \sum C_g \tag{3.3}$$

The first round simulation is described in Algorithm 3.2. The CAT tool users can set a global resistance value R_{open} for all open defects and a delay threshold t_{th} (Line 02). For all open defects in the compact set, we apply the set of two-cycle cell patterns and record the delay per combination of the open branch and cell pattern $t_{max}(b, p)$ at cell output. Only if $t_{max}(b, p)$ is larger than t_{th}, we mark the DDM value of the defect-pattern combination (*open*, *p*) as 1 (Lines 04–09). If the branch is Type 3, we calculate the delay caused by the most downstream segment open defect in Line 11. C_{min} and

C_{max} which are calculated by Equation 3.3 are the equivalent capacitances after the most downstream and the most upstream segments, respectively. If the calculated delay $t_{min}(b, p)$ is smaller than t_{th}, we record b as a suspicious branch and send it to the next round simulation.

Pattern	d1	d2?	d3?	d4
p0	1	1	0	0
p1	1	0	1	0
p2	0	0	0	0
p3	0	0	0	0

Figure 3.7: Two additional DDM columns, Columns d_2 and d_3, possibly exist if the DDM columns of two extreme open defects on a branch are as Columns d_1 and d_4.

Algorithm 3.2: OPEN-DEFECT SIMULATION

```
01: set open-defect resistance R_open and t_th;
02: for all c ∈ Cells do {
03:   suspiciousBranches_c = ∅;
04:   for all open ∈ openCompSet_c do {
05:     b = Branch(open)
06:     for all p ∈ two-cycle cell patterns_c do {
07:       delay t_max(b, p) = defectSimulation(c+open, p);
08:       if t_max(b, p) > t_th then {
09:         DDM(open, p) = 1;
10:         if b is Type 3 then {
11:           t_min(b, p) = t_max(b, p) × (C_min/C_max);
12:           if t_min(b, p) < t_th then {
13:             suspiciousBranches_c = suspiciousBranches_c ∪ b;
14: } } } } } }
```

After the second-round simulation, equivalence verification, which filters out the false positives, we can further reduce the simulation times. If only one cell pattern p causes different detection results of the two extreme opens in a branch, only two DDM columns will result from all open defects. The two DDM columns are identical except values in the p row. Therefore, it is unnecessary to simulate the full set of open defects for such a branch. If more than one cell pattern causes different detection results, we simulate every segment open in the branch to identify all different DDM columns for the best

accuracy. For example, assume that the most upstream open defect d_1 in branch b is detected by two patterns p_1 and p_2, but the most downstream open defect d_4 in b is detected by neither. If some intermediate segment open defects can be detected by either p_1 or p_2, four possible DDM columns will be caused by the open defects in b, shown as Figure 3.7. To identify the existence of *d2* and *d3* DDM columns, we run defect simulation.

The proposed algorithm is described in Algorithm 3.3. We count the number of cell patterns that cause different detection results for the two extreme open defects per suspicious branch (Lines 03–09). If more than one cell pattern causes different detection results, we perform defect simulation for all open defects to identify all different DDM columns caused by these open defects (Lines 14–18). To facilitate the downstream cell-aware ATPG, we collapse the identical DDM columns of the open defects in the same branch into one to save the downstream cell-aware ATPG time (Line 20).

Algorithm 3.3: OPEN-DEFECT EQUIVALENCE VERIFICATION AND SUPPLEMENTARY SIMULATION

```
01: for all c ∈ Cells do {
02:   for all b ∈ suspiciousBranches_c do {
// Equivalence verification for the suspicious branches
03:     patternCounter = 0;
04:     Identify the most downstream segment open open_d;
05:     for all p ∈ two-cycle cell patterns_c do {
06:       delay t_min(b, p) = defectSimulation(c+open_d, p);
07:       if t_min(b, p) < t_th then {
08:         DDM(open_d, p) = 0;
09:         patternCounter += 1; }
10:     } }
11:     if patternCounter == 1 then {
12:       Add DDM column of open_d to DDM;
13:     }
// Supplementary simulation
14:     if patternCounter > 1 then {
15:       for all p ∈ two-cycle cell patterns_c do {
16:         for all segm(i, j) ∈ b do {
17:           d = R_open+segm(i, j);
18:           DDM(d, p) = defect_simulation(c+d, p);
19:       } }
20:     Collapse identical DDM columns into one for opens on b;
21:     }
22: } }
```

3.3 Short-Defect Identification and Simulation

This section describes how we handle the full set of short defects to reduce the simulation time. Section 3.3.1 analyzes the electrical behaviors of short defects with the aim to identify their equivalence, which is the basis of short-defect collapsing and defect equivalence verification. The algorithm of open-defect collapsing is also described. Section 3.3.2 introduces the three-round simulations for short defects.

3.3.1 Short-Defect Equivalence and Collapsing

A short defect can occur between any two nodes from two disjoint nets a and b and cause the whole net pair to be shorted together. Figure 3.8 shows a partial schematic of a full adder cell in Cadence's GPDK045 library. To detect a short defect between a and b, we should first activate the short defect by assigning opposite logic values on a and b. As each cell-internal net is driven by either a pull-up or a pull-down network, logic 1 on a net is obtained by switching on the pull-up network to connect the net to the power net V_{DD}, while logic 0 is obtained from the ground net V_{SS} through the pull-down network. When a static signal is applied on a net, no current flows on the net as the logic signal is used to control downstream transistor gates that are considered high-impedance terminals. In consequence, the short defect is a conductive path from V_{DD} to V_{SS}, shown in Figure 3.8. If the interconnects are ideal, i.e. without any parasitic resistance, the logic values on both nets result from the voltage division between the transistor networks driving the two shorted nets, assuming the short-defect resistance is small and can be ignored. If the short-defect resistance increases, the voltage difference between the two shorted nets, which is equal to the voltage drop on the short-defect resistor, also increases. As long as the interpreted logic values on the two shorted nets are the same, the analysis is still valid. No matter which two nodes from net a respectively b are shorted and how big R_{short} is, the voltage value at the short location is fixed. All short defects between the two nets cause the same electrical effect and hence are equivalent. Multiple cell patterns can activate a short defect. In Figure 3.8, logic 1 on net a and logic 0 on net b can be obtained by three stimulus patterns: {A=0,B=1,C=1; A=1,B=0,C=1; A=0,B=0,C=1}. Different stimulus patterns switch on different transistors of which resistances determine the voltage division in the path from V_{DD} to V_{SS}.

In reality, the interconnects are not ideal, but have some parasitic resistance. Considering the parasitic resistance, per stimulus pattern, the voltage values at different short-defect locations are slightly different. We show a short defect model including the extracted parasitic resistors on nets for the circuit of Figure 3.8, shown in Figure 3.9. To make the analysis more easily understood, we assume R_{short} is approximately 0Ω, i.e. the voltage

at the two shorted nodes is equal. The numbers of nodes on nets a and b are *na* and *nb*, respectively. The short defect can be between node a_i on net *a* and node b_j on net *b* if $C(a_i, b_j)$ exists in the extracted DSPF netlist, where $i \in [0..na)$ and $j \in [0..nb)$. The logic value at a short-defect location is determined by the voltage division between two resistances on the two sides of the location. Even though the voltage can be any value between V_{DD} and V_{SS}, the transistors driven by this voltage always interpret it into logic 0 or 1. In Figure 3.9, the two resistances are ($R_{pull\text{-}a}+R_a$) and ($R_{pull\text{-}b}+R_b$), where $R_{pull\text{-}a}$ and $R_{pull\text{-}b}$ are the resistances of switched-on transistors in the pull networks driving nets a and b, respectively. R_a and R_b are the parasitic resistances of net a and b involved in the conductive path from V_{DD} to V_{SS} due to the short defect. R_a is the sum of resistances from node a_0 till node a_i which is the short location node. Similarly, R_b is the sum of resistances from node b_0 till the short location b_j. A cell pattern causing opposite logic values on nets a and b determines the conducting transistors in the pull-up and pull-down networks, and therefore determines the network resistances $R_{pull\text{-}a}$ and $R_{pull\text{-}b}$. R_a and R_b are determined by the short-defect location.

Given a cell pattern p activating a short defect between nets a and b, the voltages V_a and V_b at the shorted node a_i and b_j vary with the locations of a_i and b_j, as their locations affect R_a and R_b. Typically, the parasitic resistance of interconnects is small, and hence the voltage drop on parasitic resistors is much smaller than that on transistors. As the source-drain voltage of transistors does not significantly change with the variation of the short location, we assume the transistors in the shorted path can be seen as linear resistors [Rabaey et al., 2003]. With this assumption, a larger R_a and a smaller R_b increase the voltage drop on net a. This increases the probability that the interpreted logic value at node a_i will be different from the defect-free value. When R_a is maximized and R_b is minimized by shorting nodes a_{na-1} and b_0, shown in Figure 3.10(a), the voltage di-

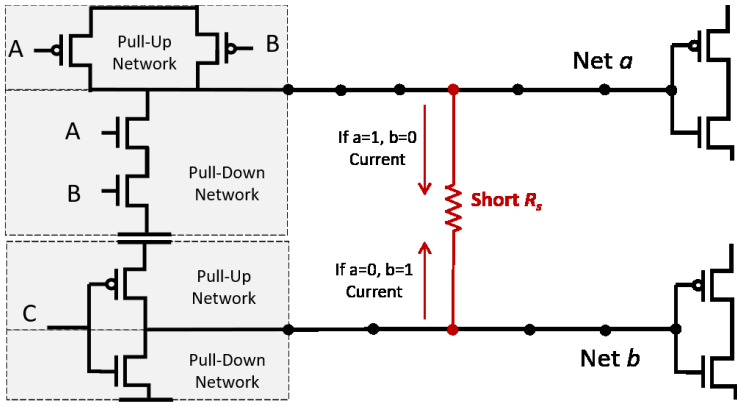


Figure 3.8: Short defects between nets without parasitic resistance.

vision is determined by the ratio of $(R_{pull\text{-}a} + max(R_a))$ to $R_{pull\text{-}b}$. If a pull-up network drives net a and a pull-down network drives net b, the short defect in Figure 3.10(a) leads to the minimum V_a and V_b which is most probably interpreted as logic 0. This means the logic value on net a is forced to change from logic 1 to 0. If a pull-down network drives net a and a pull-up network drives net b, the voltage drop on net a is maximized and the voltage drop on net b is minimized to zero, and the short defect in Figure 3.10(a) leads to the maximum V_a and V_b. This increases the probability that the logic value on a and b is 1, and a is still dominated by b. Similarly, Figure 3.10(b) shows the short-defect location which causes the highest probability that net a dominates net b. In summary, between net pair (a, b) the two diagonal short defects shown in Figure 3.10(a) and Figure 3.10(b) determine the range of voltages at short-defect locations for every cell pattern activating the short defects between nets a and b. All short defects between a and b are equivalent, if for all cell patterns the voltage values at the two diagonal short-defect locations are interpreted as identical logic values. Typically, transistor network resistances are much larger than the parasitic resistances, and therefore varying the short-defect location does not change the logic value on the shorted nets change. Thus, all short defects are equivalent. However, between some net pairs, varying the short-defect location results in some non-equivalent defects that should be identified. For these net pairs, we perform additional simulations to determine the complete DDM columns caused by all short defects between them.

Power net V_{DD} and ground net V_{SS} are special nets, as they are voltage sources, and no transistor network drives them. Assuming a short defect occurs between V_{DD}/V_{SS} and some other net, the voltage at the short-defect location is determined by the voltage drop on the parasitic resistance of V_{DD}/V_{SS}. Typically, the parasitic resistance on V_{DD}

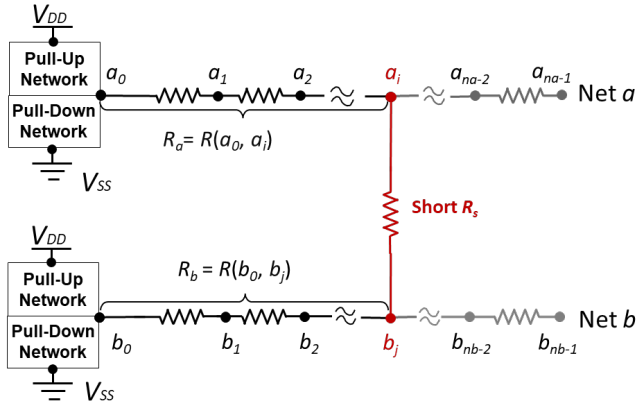


Figure 3.9: Short defects between nets with parasitic resistance.

or V_{SS} is up to several tens of ohms, while the transistor network resistance is at least several hundred ohms. The voltage drop on V_{DD} or V_{SS} is much smaller than that on the other side of the short defect, and a change of short-defect location cannot lead to a drastic voltage variation at the short-defect location. Therefore, the voltage at the short-defect location is always close to V_{DD} or V_{SS}. Thus, V_{DD} or V_{SS} always dominates the net driven by cell transistors. Therefore, all short defects between a net driven by transistors and a power rail are equivalent. For such net pairs, it is sufficient to simulate only one short defect to guarantee test quality.

Based on the analysis of short-defect equivalence, for many net pairs, the short defects are equivalent. To minimize the number of defects to be simulated and maintain the test quality, we propose the following four-step flow, shown in Figure 3.1. In Step 1, we collapse the full set of short defects into a compact set by identifying only one short-defect location and simulate the compact set. This location should meet two conditions: (1) the location is based on an extracted parasitic capacitor and (2) compared with the other extracted capacitors, the node pair of this capacitor causes that on one net the maximum parasitic R while on the other net the minimum parasitic R are involved into the shorted path, and therefore results in either boundary of the voltage variation due to the short location. In Step 2, the compact set of short defects is simulated and we identify the net pairs that possibly have non-equivalent short defects based on the simulation results. We use analog simulation to verify if short defects between these suspicious net pairs are equivalent and filter out the false positives in Step 3. In Step 4, we simulate the full set of short defects for the net pairs with non-equivalent short defects to restore the test quality.

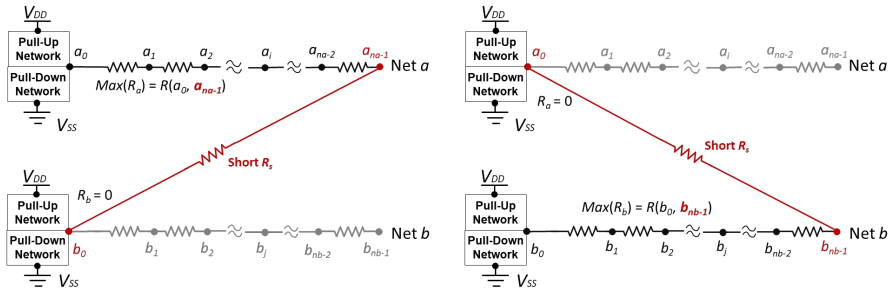


Figure 3.10: Two extreme short-defect locations lead to the highest probability that (a) net b dominates and (b) net a dominates.

Algorithm 3.4: SHORT-DEFECT COLLAPSING

```
01: for all c ∈ Cells do {
02:   shortCompSet_c = {};
03:   for all a, b ∈ Nets_c with a ≠ b do {
04:     if [(a, b)] ∉ shortCompSet_c then {
05:       mi = []; mj = []; R_mi = 0; R_mj = 0
06:       for all i ∈ a and j ∈ b do {
07:         if (i, j) ∈ shortFullSet_c then {
08:           if (R_i − R_j) > (R_mi − R_mj) then {
09:             R_mi = R_i; R_mj = R_j; mi = [i]; mj = [j];
10:       shortCompSet_c[(a, b)] = (mi, mj);
11:   } } } } }
12:   if ¬ disableTrTerminalShorts then {
13:     for all transistors t in c do {
14:       for all transistor terminal pairs (tm_1, tm_2) ∈ TrTerminalShortSet do {
15:         if [(net(tm_1),(net(tm_2))] ∉ shortCompSet_c then {
16:           shortCompSet_c[(net(tm_1),(net(tm_2))] = (tm_1, tm_2);
17:         else {
18:           i = shortCompSet_c[net(tm_1),(net(tm_2))][0];
19:           j = shortCompSet_c[net(tm_1),(net(tm_2))][1];
20:           if R_tm1 − R_tm2 > R_i − R_j then {
21:             shortCompSet_c[(net(tm_1),(net(tm_2))] = (tm_1, tm_2);
22: } } } } } } }
```

The short-defect collapsing algorithm is described in Algorithm 3.4. We use dictionary $shortCompSet_c$ to store the collapsed short-defect location per net pair (Line 02). The keys of dictionary $shortCompSet_c$ are net pair tuples, and the value corresponding to each key is a tuple of two nodes identified as the short-defect location. Subsequently, for each net pair $a, b \in Nets_c$ which does not have a short identified yet (Line 05), we identify the node pair $i \in a$ and $j \in b$ from the full set of short-defect locations and this node pair should lead to the maximum difference of parasitic resistances on nets a and b that are involved in the shorted path (Lines 06–10). For transistors, two reasons cause two nets connected to two terminals of the same transistor to have no short defect identified: (1) either or both transistor terminals connect to virtual nets and (2) all extracted capacitors between the two nets are filtered out by a user-defined capacitance threshold, because the small capacitances indicate low probabilities of a short defect occurring. Therefore, for all transistors, if the user does not disable the transistor terminal shorts, only the transistor terminal pairs between which there is not already a short between the two corresponding nets are added to the compact set (Lines 12–16). If the transistor terminal leads to a larger difference in the involved parasitic resistances, we replace the

interconnect short location with the transistor-terminal pair (Lines 17–21).

3.3.2 Short-Defect Simulation and Equivalence Verification

Defect simulation is performed with only one short defect per net pair. To verify the test equivalence of short defects, we check per combination of short defect and cell pattern, voltage value V_p at the short-defect location and current I_p in the conductive path from $\mathrm{V_{DD}}$ to $\mathrm{V_{SS}}$.

With the voltage and current values at the location of each simulated short defect, we calculate V_p/I_p which is the sum of the resistance of the pull-down network and the parasitic resistance on the corresponding driven net. Similarly, $(V_{DD} - V_p)/I_p$ is pull-up network resistance plus the parasitic resistance on the corresponding driven net. The involved parasitic resistance on two shorted nets is calculated by processing the extracted DSPF netlist. We calculate the transistor-network resistances by subtracting the parasitic resistances in V_p/I_p and $(V_{DD} - V_p)/I_p$. Considering the effect of the short-defect location, we classify the relationship between the transistor and parasitic resistances into four cases per cell pattern p that activates the short defect, shown in Figure 3.11.

- **Case 1**: $(R_{pull-a}(p)/R_{pull-b}(p) > 3/2) \wedge (R_{pull-a}(p) > 10 \times max(R_a)) \wedge (R_{pull-b}(p) > 10 \times max(R_b))$
  If no parasitic resistance is involved in the conductive path caused by a short defect, $R_{pull-a}/R_{pull-b} > 3/2$ results in more than 60% voltage drop on the transistor network driving net a. If a pull-up network drives net a, the voltage at the short location V_p is less than $40\%\mathrm{V_{DD}}$. As a result, the logic values on nets a and b are 0. If a pull-down network drives net a, V_p is larger than $60\%\mathrm{V_{DD}}$, and the logic values on a and

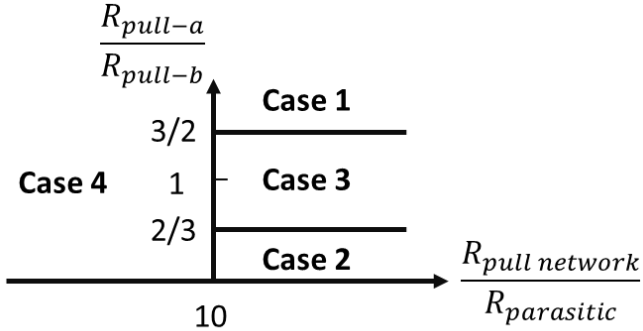


Figure 3.11: Area of four cases.

b are interpreted as logic 1. Therefore, net b dominates net a. Furthermore, transistor resistances are much larger than the maximum of R_a and R_b. The variations of R_a and R_b due to the short-defect location cannot cause the voltage at the short-defect location to vary between logic 0 and 1. The effect of short-defect location can be ignored, and hence all short defects between a and b are equivalent for p.

- **Case 2**: $(R_{pull-a}(p)/R_{pull-b}(p) < 2/3) \wedge (R_{pull-a}(p) > 10 \times max(R_a)) \wedge (R_{pull-b}(p) > 10 \times max(R_b))$
  Case 2 is similar to Case 1. The voltage at the short-defect location is also either smaller than $40\%\mathrm{V_{DD}}$ or larger than $60\%\mathrm{V_{DD}}$, and the effect of short-defect location can be ignored. Thus, all short defects are equivalent for pattern p, but in this case, net a dominates net b.
- **Case 3**: $2/3 \leqslant R_{pull-a}(p)/R_{pull-b}(p) \leqslant 3/2 \wedge (R_{pull-a}(p) > 10 \times max(R_a)) \wedge (R_{pull-b}(p) > 10 \times max(R_b))$
  When no parasitic resistance is involved in the conductive path caused by a short defect, the voltage value at the short-defect location is in the range of [$40\%\mathrm{V_{DD}}$, $60\%\mathrm{V_{DD}}$]. As the threshold voltage between logic 0 and 1 for the CMOS transistors is in this range, a small voltage variation can change the interpreted logic value. Therefore, the variations of R_a and R_b may cause different fault effects. In this case, two possible groups of equivalent short defects exist between the net pair (a, b). One group causes the logic values on both nets to be 1, while the other group causes both nets' logic values to be 0.
- **Case 4**: $(R_{pull-a}(p) \leqslant 10 \times max(R_a)) \vee (R_{pull-b}(p) \leqslant 10 \times max(R_b)) \vee (R_{pull-b}(p) \leqslant 10 \times max(R_a)) \vee (R_{pull-a}(p) \leqslant 10 \times max(R_b))$
  In standard cells with high drive strength, multiple parallel transistors are controlled by the same input signal. When these parallel transistors are switched on simultaneously, the signal on their driven net is pulled up or down by these parallel transistors together, and hence the drive strength increases. However, resistances of the transistor networks offering high drive strength decrease due to the parallel conducted transistors. If the transistor-network resistances are comparable to the parasitic resistance, the short-defect location may influence the logic value on the shorted nets a and b. For the net pairs of which parasitic resistances are larger than 10% transistor-network resistance, we check if the two diagonal short defects are equivalent by analog simulation.

If Case 1 or 2 happens for a cell pattern, all short defects between the corresponding net pair are equivalent. Otherwise, i.e. for Cases 3 and 4, defect equivalence verification is required to maintain the test quality. The process is described in Algorithm 3.5. For each short defect in the compact set, we record not only the *DDM* value but also the voltage and current values (Line 07). Afterwards, we calculate the pull network resistances and identify the suspicious net pairs and the corresponding cell patterns that could lead to different detection results of the short defects between these net pairs (Line 08–21).

We use a dictionary to store the combinations of suspicious net pair and cell patterns (Line 05). The keys of the dictionary are net pairs and the value corresponding to each key is the set of cell patterns for which we need to verify the equivalence of all short defects between the key net pair.

Algorithm 3.5: SHORT-DEFECT SIMULATION

```
01: set short-defect resistance R_short;
02: for all c ∈ Cells do {
03:   for all (a, b) ∈ shortCompSet_c do {
04:     s = (shortCompSet_c[(a, b)], R_short);
05:     suspiciousNetPairs_c[(a, b)] = ∅;
06:     for all p ∈ one-cycle cell patterns_c do {
07:       Voltage(s, p), Current(s, p), DDM(s, p) = Simulation(c + s, p);
08:       if a is driven by a pull-down network ∧ b is driven by a pull-up
          network then {
09:         R_pull-a(p) = (Voltage(s, p)/Current(s, p)) - R_a;
10:         R_pull-b(p) = (V_DD − Voltage(s, p))/I_p - R_b - R_short;
11:       }
12:       else if a is driven by a pull-up network ∧ b is driven by a pull-down
          network then {
13:         R_pull-a(p) = (V_DD − Voltage(s, p))/I_p - R_a - R_short;
14:         R_pull-b(p) = (Voltage(s, p)/Current(s, p)) - R_b;
15:       }
16:       else {continue}; // a or b is driven by the power or ground net.
17:       if (R_pull-a(p)/R_pull-b(p) < 2/3 ∨ R_pull-a(p)/R_pull-b(p) > 3/2)
              ∧ (R_pull-a(p) > 10 × max(R_a) ∧ R_pull-b(p) > 10 × max(R_b)) then {
18:         all short defects between nets a and b are equivalent for p;
19:       }
20:       else {
21:         suspiciousNetPairs_c[(a, b)] = suspiciousNetPairs_c[(a, b)] ∪ {p};
22: } } } }
```

The equivalence identification and supplementary simulation are presented in Algorithm 3.6. The identification of net pairs that are suspicious, i.e., may have non-equivalent short defects includes some false positives. We should first filter out the false positives by using analog simulations. The two diagonal short defects determine the extreme voltages on the shorted nets per cell pattern. If they are equivalent, all short defects between the shorted nets are equivalent. Therefore, we perform analog simulation for the other diagonal short defect and determine the net pairs with non-equivalent short defects out of the suspicious net pairs (Line 02–12). If for only one cell pattern the short defects

are non-equivalent, we derive only two different DDM columns in which the entries are all identical except in pattern p row. (Lines 13–19). Otherwise, we simulate all short defects only with the identified cell patterns in *determinedNetPairs* to get all different DDM columns (Lines 20–25). To facilitate the downstream ATPG step, we collapse all identical DDM columns into one (Line 26).

Algorithm 3.6: SHORT-DEFECT EQUIVALENCE VERIFICATION AND SUPPLEMENTARY SIMULATION

```
01: for all c ∈ Cells do {
// Equivalence verification
02:   for all (a, b) in suspiciousNetPairs_c do {
03:     determinedNetPairs[(a, b)] = ∅;
04:     s = (shortCompSet_c[(a, b)], R_short);
05:     identify the other diagonal node pair (i ∈ a, j ∈ b) with
        maximum (R_b − R_a);
06:     inject short defect s' with resistance R_short between (i, j);
07:     for all p in suspiciousNetPairs_c[(a, b)] do {
08:       DDM(s', p) = Simulation(c + s', p);
09:       if DDM(s, p) ≠ DDM(s', p) then {
10:         short defects between a and b are non-equivalent for p;
11:         determinedNetPairs[(a, b)] = determinedNetPairs[(a, b)] ∪ {p};
12:     } }
13:     if |determinedNetPairs[(a, b)]| = 1 then {
14:       for all pattern p ∈ one-cycle cell patterns_c do {
15:         if p ∈ determinedNetPairs[(a, b)] then {
16:           DDM(s', p) = 1− DDM(s, p);}
17:         else {DDM(s', p) = DDM(s, p); }
18:       add new DDM columns of s' to DDM;
19:     } }
// Supplementary Simulation
20:     if |determinedNetPairs[(a, b)]| > 1 then {
21:       for all short s', with s' ≠ s, between a and b, do {
22:         for all pattern p ∈ one-cycle cell patterns_c do {
23:           if p ∈ determinedNetPairs[(a, b)] then {
24:             DDM(s', p) = Simulation(c + s', p); }
25:           else {DDM(s', p) = DDM(s, p)};
26:       collapse all identical DDM columns into one column;
27: } } } } }
```

3.4 Experimental Results

The experiment results reported in this thesis are based on the 476 cells in Cadence's GPDK045 45nm library [Cadence Design Systems, 2014]. This library uses a planar CMOS technology and all cells contain four layers: n-diffusion, p-diffusion, poly, and metal-1 layer. The EDA tool versions used in our experiments are Cadence Pegasus v18.1, Quantus Extraction Solution v18.1, and Modus v20.1. For the full set DLI, we use the default user settings which include all possible defect locations for the best test quality. For defect characterization, we set the open-defect resistance to 1GΩ and the short-defect resistance to 0.001Ω. The tolerated delay threshold value is set to 10ns.

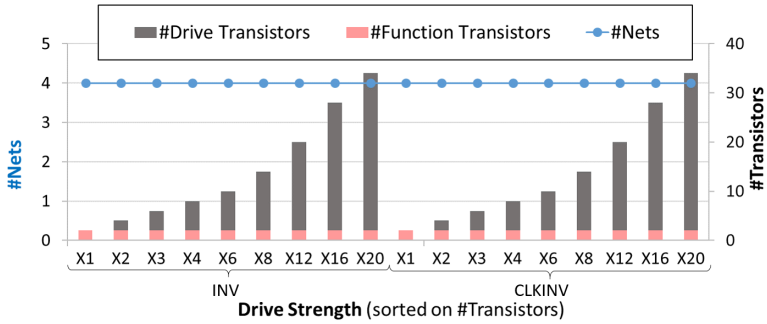
3.4.1 Overview of GPDK045 CMOS Library

A logic cell with base drive strength that is denoted by 'D1' or 'X1' in the cell name contains only transistors that implement the cell's logic function. We refer to these transistors as *function transistors*. Designers increase the drive strength of cells by adding so-called *drive transistors* in parallel with the function transistors driving the cell outputs. Function and drive transistors are both switched on or off by the same input signal and contribute to pull up or pull down the output simultaneously, which increases the drive strength. For a group of cells with the same logic function, the variation of drive strength influences the number of transistors but not the number of nets, thus the number of short-defect locations in the compact set is independent of the drive strength. For example, Figure 3.12 shows the drive strengths and the number of transistors and nets in all GPDK045 inverter and multiplexer 2-to-1 cells. The number of nets always stays the same for all inverter or multiplexer cells, but the number of transistors increases with the drive strength.

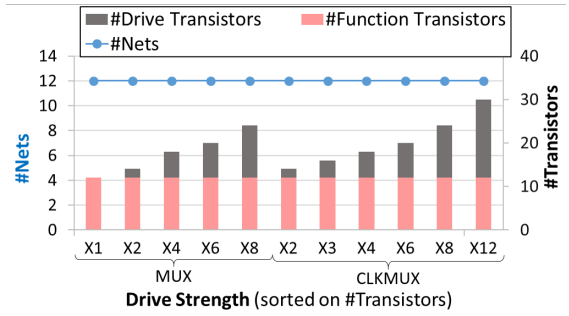
The GPDK045 cell information is shown in Figure 3.13. The total 476 cells consist of 324 combinational and 152 sequential cells. The most complex library cells (AOI33, MUX4, SDFFSR, etc.) have six inputs, and require the most cell patterns for the simulation. We use the exhaustive set of one-cycle cell patterns and a limited set of two-cycle patterns for the defect characterization. The identification of the two-cycle cell patterns is explained in Section 2.6.2.

3.4.2 Experimental Results of Example Cell AND2X1

First, we illustrate our approach with cell AND2X1. The AND2X1 cell has two inputs, A and B, one output Y, and performs a logic-AND function. Three I/Os, power and



(a) GPDK045 inverter cells.



(b) GPDK045 multiplexer 2-to-1 cells.

Figure 3.12: The number of transistors but not nets increases with the drive strength.

ground connections, and six transistors with four terminals give a total of 29 terminals for this cell. As shown in Figure 3.14(a), these terminals are interconnected by seven nets: $\{A, B, Y, V_{\mathrm{DD}}, V_{\mathrm{SS}}, n1, n2\}$.

Figure 3.14(b), shows the layout of AND2X1. The PEX tool partitions the seven nets into 43 segments excluding eight virtual segments, each with its own parasitic resistor. Figure 3.14(c) illustrates the PEX partitioning of nets into segments for Net A, which interconnects input terminal A and the gate terminals of Transistors $Mp0$ and $Mn1$. PEX introduces four internal nodes on Net A. Nodes $A\#9$ and $A\#4$ mark the transition points between metal-1 and poly-silicon to the contact between these two layers, respectively. At node $A\#3$, the net forks out to the PMOS and NMOS transistors. Node $A\#5$ is introduced to mark the 90° bend point in the 'L'-shape. The dangling segments at the extreme ends of the poly-silicon and metal-1 structures are not reported. Example net A consists of six segments, partitioned over three branches: $\{A, A\#9, A\#4, A\#3\}$,

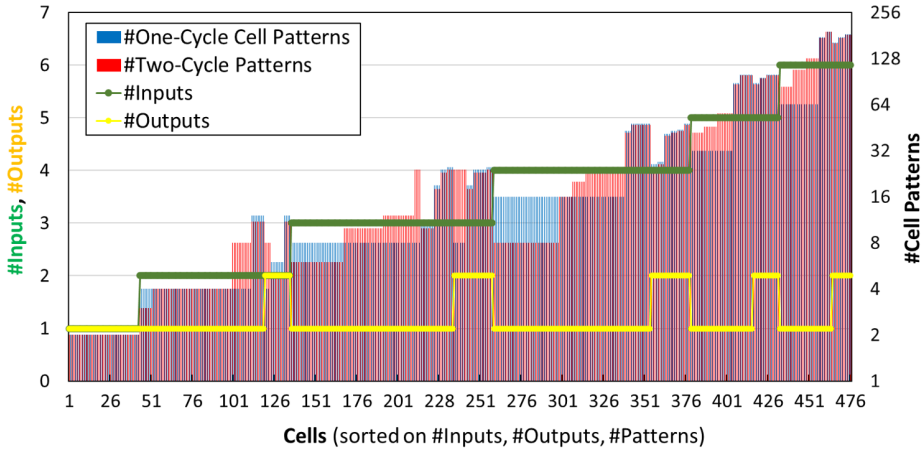
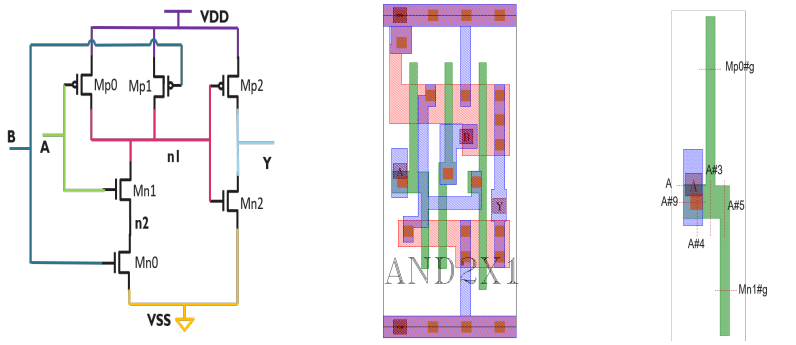


Figure 3.13: The number of library cells' inputs, outputs, and test patterns.

$\{A\#3, Mp0\#g\}$, and $\{A\#3, A\#5, Mn1\#g\}$. For each of the three branches, we identify the segment on which an open defect is the most likely to happen. Opens on the latter two branches are equivalent to transistor gate-terminal opens on $Mp0\#g$ and $Mn1\#g$. Therefore, these two gate-terminal opens are not selected into the compact set.



(a) Schematic. (b) Layout. (c) Nodes on Net A.

Figure 3.14: GPDK045 library cell AND2X1.

For the complete cell AND2X1, identifying open-defect locations on *all* segments and on *all* gate, source, and drain transistor terminals results in 43+18 = 61 open-defect locations. However, AND2X1 has 20 branches; we collapse the open-defect locations

Pattern	Two-Cycle Detection: Delay Matrix (unit: ns)											
AB;AB/Y	Defect-Free	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10	d11
p0 = 01;11/H	0.050	49.19		33.09	62.46		116.16	503.76				799.67
p1 = 10;11/H	0.052		54.30	33.09		60.26	116.07		31.16		607.03	843.77
p2 = 11;01/L	0.037			62.99	70.01		29.18	479.99				834.70
p3 = 11;10/L	0.039			62.95		71.16	32.11			26.20	593.98	880.41

Table 3.1: Delays caused by the most upstream segment open per branch.

Pattern	Two-Cycle Detection: Delay Matrix (unit: ns)											
AB;AB/Y	Defect-Free	d1′	d2′	d3′	d4′	d5′	d6′	d7′	d8′	d9′	d10′	d11′
p1 = 01;11/H	0.050	39.61		33.09	62.46		116.16	395.23				666.46
p2 = 10;11/H	0.052		36.83	33.09		60.26	116.07		31.16		459.44	703.21
p3 = 11;01/L	0.037			62.99	70.01		29.18	376.59				695.65
p4 = 11;10/L	0.039			62.95		71.16	32.11			26.20	449.56	733.75

Table 3.2: Calculated delays caused by the most downstream segment open per branch.

on 43 segments into on 20 branches. Afterwards, only seven transistor-terminal opens are not on any physical branch. So, only 20 branches and three transistor-terminal opens out of 61 (= 37.7%) open-defect locations are selected for the compact set. Among the 20 branch opens, nine opens cause either no delay or a delay longer than 1μs (= 1000ns) for all cell patterns. As opens on these nine branches cause a delay much larger than the tolerated delay threshold defined by the user, and the load capacitance variation on a branch cannot be 1000×, all open defects on these branches will cause delay larger than 1ns. They are detected by the same patterns, and thus are equivalent. The other eleven open defects causing a delay smaller than 1μs and larger than 1ns are shown in Table 3.1. Based on the load capacitance variation due to the open-defect location per branch, we calculate the delay caused by the most downstream segment open on each of the eleven branches, shown in Table 3.2. All calculated delays in Table 3.2 are larger than 10ns. Therefore, for all branches, the most upstream and the most downstream segment opens are detected by the same set of cell patterns. All opens on the branches are equivalent, and no additional simulation is to be performed.

For Branches on which open defects {d3, d3′}, {d4, d4′}, {d5, d5′}, {d6, d6′}, {d8, d8′}, {d9, d9′} are located, the delays caused by the most upstream and the most downstream segment opens are identical. The reason can be that either the branch contains only one segment, or the load capacitance variation caused by open-defect location is too small to impact the delay.

Next, we consider short-defect locations. Cell AND2X1 has 58 nodes, which make for $\binom{58}{2}$ = 1,653 node pairs. Subtracting the 257 node pairs for which both nodes are part of the same net leaves 1,396 node pairs between disjoint nets; this is the theoretical

maximum for the number of short-defect locations of this cell. The PEX tool extracts only 745 parasitic capacitors above its user-specified PEX minimum of 10^{-25} F. The DLI function considers at most one short-defect location per net pair. Cell AND2X1 has six physical nets, except net *n2* connecting $Mn0\#d$ and $Mn1\#s$ between which there is no real physical interconnection but just a shared diffusion area. Therefore, we consider $\binom{6}{2}$ = 15 net pairs. As they all have extracted parasitic capacitors, for each one we identify one representative short-defect location by selecting the capacitor with the maximum value. For each transistor, identifying short-defect locations between three terminal pairs would result in 18 shorts. However, some transistor-terminal short locations are between the same net pairs, and hence are equivalent. Only 15 terminal shorts are non-equivalent to each other and ten of them are equivalent to the net-pair shorts already identified based on the parasitic capacitors. Therefore, we only need to add five more transistor-terminal short locations into the compact set. Furthermore, three of these shorts (A-Y, B-Y, and A-B) are equivalent to the *port-shorts* at the cell boundary. In total, 20 of the 21 potential net-pair short locations are considered realistic in our approach; this constitutes a reduction of 97.4% from the 745+24 = 769 possible short locations based on the extracted netlist. The corresponding simulation time is reduced from 74 to only 7 seconds, which is a 90.5% reduction. The 20 identified cell-internal short-defect locations also cover the six stuck-at faults at the cell boundary, i.e., either stuck-at-zero or -one at ports A, B, and Y.

Table 3.3 shows the voltages at short-defect locations given by the simulation. The power voltage value is set to 1.1v based on the library Liberty files. Cell patterns are in the first column. The second row gives the names of net pairs between which we inject a diagonal short defect. The shorted net pairs of which one net is V_{DD} or V_{SS} are not listed in Table 3.3 as V_{DD} or V_{SS} always strongly dominates the other net. And it is not necessary to verify the short-defect equivalence for these net pairs. The highlighted items in Table 3.3 indicate the combinations of net pair and cell pattern that may have multiple detection results when changing the short-defect location, because the voltage value at the short-defect location is close to the threshold of logic 1. For all the other net pairs, the voltage values at short-defect locations are strong logic '1' or '0', and their parasitic resistances are much smaller than the driving transistor networks. Therefore, the short-defect location does not impact the detection results. Only for net pair *A*-*B*, we simulate the other diagonal short defect with only cell patterns '01/L' and '10/L'. The resulting DDM columns of the two diagonal shorts are identical, so all short defects between *A* and *B* will cause identical DDM columns, i.e. are equivalent. The test quality is the same as that based on the full set of defects. We do not need to perform any additional simulations.

Pattern	One-Cycle Detection: Voltage Matrix (unit: v)					
AB/Y	A-B	A-Y	B-Y	A-n1	B-n1	Y-n1
p0 = 00/L	0	0	0	0.16	0.15	0.89
p1 = 01/L	0.48	0	0.89	0.08	1.1	0.81
p2 = 10/L	0.47	0.87	0	1.1	0.07	0.81
p3 = 11/H	1.1	1.1	1.1	1.04	1.05	0.34

Table 3.3: Calculated voltage values at one diagonal short-defect location.

3.4.3 Experimental Results of GPDK045 Library

The PEX run on 476 cells of the GPDK045 library [Cadence Design Systems, 2014] resulted in 12,234 transistors, 102,218 extracted net segments with parasitic resistors, and 2,819,369 extracted parasitic capacitors, and took 19.1 hours. Note that there are 27.6× more parasitic capacitors extracted than parasitic resistors. The simulation time for short-defect characterization dominates the total time cost.

Assigning open defects to all net segments and to three terminals for each transistor results in 3 × 12,234 + 102,218 = 138,920 open-defect locations in the full set of which 59,597 open defects are selected into the compact set for the downstream defect characterization. This constitutes a reduction of 57.1%. The DLI results per library cell for opens are shown in Figure 3.15.

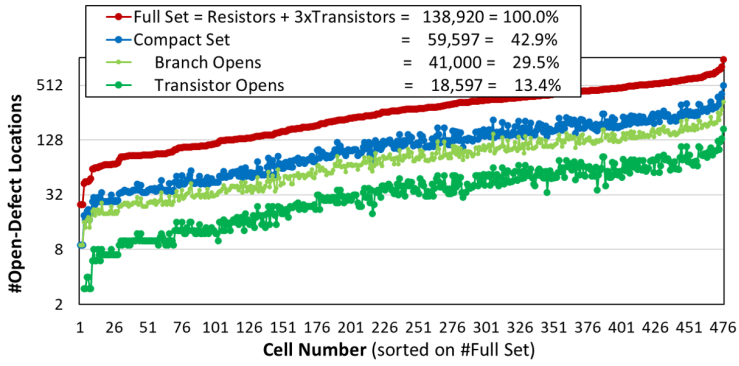


Figure 3.15: Full vs. compact sets of open-defect locations in GPDK045 library cells and the division of the latter into branch and transistor opens.

The simulation time of the compact set is 133.3 hours (5.6 days), which indicates the simulation based on the full set would take around 350.9 hours (14.6 days), assuming all defect-pattern tuples for each cell take identical simulation time. The simulation for

total 324 combinational cells only takes 26.2 hours, but the simulation time of total 152 sequential cells is 107.1 hours. The simulation for sequential cells dominates the total time cost of defect characterization. For the total 41,000 branches, 57.0% are branches of which the most upstream segment opens are detected. We verify the defect equivalence for the 57.0% branches. The other 43.0% branch opens cause delay smaller than the detectable threshold 10ns, and hence are non-detectable. As the most upstream segment opens on the 43.0% branches cause delay smaller than 10ns, all the other opens on their branches cause delay also smaller than 10ns. All opens on the 43.0% are non-detectable and equivalent. For each branch of the 57.0% branches, if the most upstream segment open causes delay size larger than the tolerated delay threshold 10ns and the most downstream segment open leads to a delay size smaller than 10ns, not all segment opens on this branch are equivalent. Before calculating the delay sizes of the most downstream opens, we filter out Type 3 branches on which open defects at different locations may cause different detection results. 37.6% branch opens cause a delay larger than 1μs for all required cell patterns, which is 100$\times$ tolerated delay threshold 10ns. On these branches, it is not possible that the delay variation is larger than 100$\times$, and therefore all opens are detectable by the same set of cell patterns, i.e. all opens are equivalent. For the remaining 57.0% - 37.6% = 20.4% branches, we classify them into four types according to the discussion in Section 3.2, shown in Figure 3.16. Only 5.9% (= 2,433) of Type 3 branches require the test quality verification.

For all 2,433 Type 3 branches, we calculate the delay size resulting from the most downstream segment open for each pattern. Figure 3.17 shows per branch, only the minimum delays caused by the most upstream and downstream segment opens by applying all cell patterns. The delay calculation is based on the assumption that all neighboring nets have an opposite transition to the open nets. This causes the calculated delay due to the most downstream open defect is always smaller or equal to the accurate delay. If the calculated delay is larger than 10ns, the accurate delay must be larger than 10ns also. The results show that on no branch, the most upstream and downstream segment opens cause different detection results. All segment opens on every branch are equivalent. The branch classification and delay size calculation for the Type 3 branches took $\sim$1.6 hours in total. In summary, we reduce the open-defect simulation time by (1 - ((133.3 + 1.6)350.9)) $\times$ 100% = 61.6%, maintaining the best test quality.

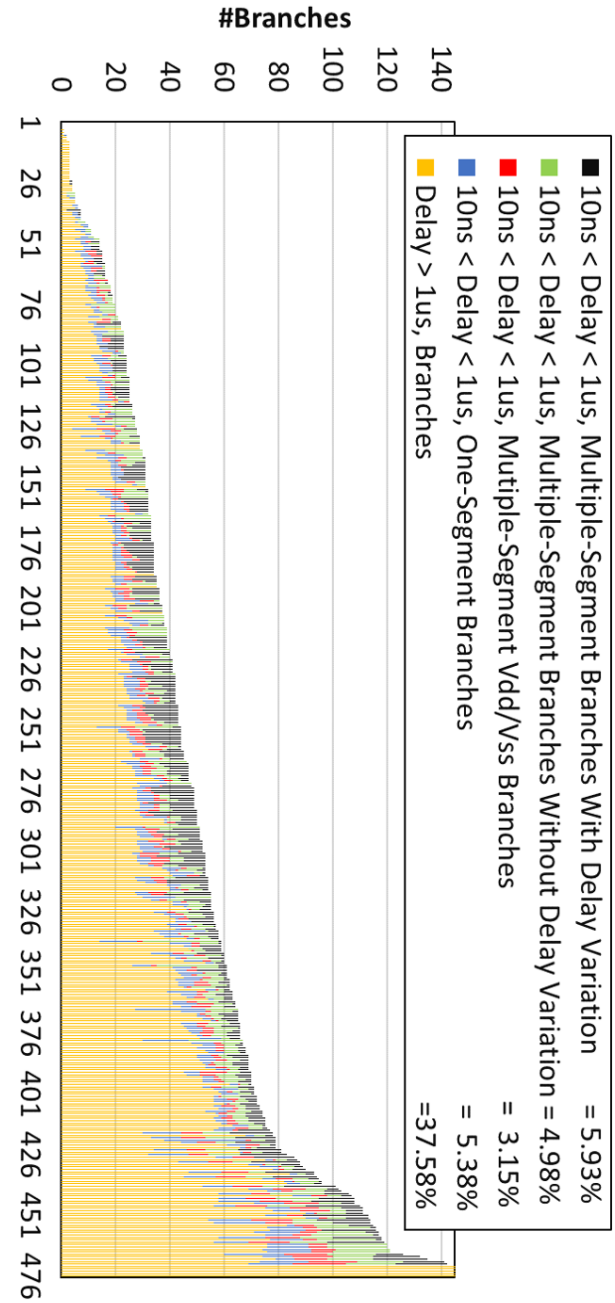


Figure 3.16: Minimum delays caused by the most-upstream and most-downstream segment opens.

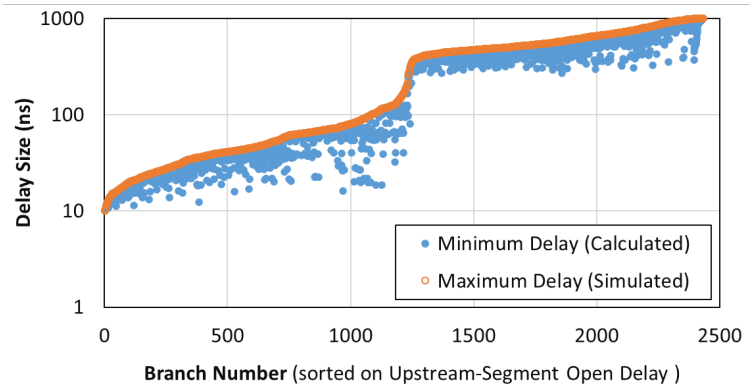


Figure 3.17: Minimum delays caused by the most-upstream and most-downstream segment opens.

The 476 library cells have a total sum of 14,269,038 node pairs on disjoint nets. The PEX tool only extracts 2,819,369 parasitic capacitors (excluding self-capacitances). If we assign shorts on all parasitic C and three short defects per transistor between its transistor terminals, we identify a full set of 2,819,369 + 3 $\times$ 12,234 = 2,856,071 short-defect locations. We consider one diagonal short defect per net pair. The library cells together have 30,490 net pairs, for which only 21,674 net pairs have extracted parasitic capacitors. We identify 21,674 net-pair shorts in the compact set. Subsequently, we add additional 8,564 transistor-terminal shorts that are between non-physical interconnects but modeling transistor-internal defects. The compact set reduces the full set of short locations with 1 - (30,238/(2,819,369 + 3 $\times$ 12,234)) $\times$ 100% = 98.9%. Figure 3.18 shows per library cell the DLI results for shorts. Unlike what was claimed by [Hapke et al., 2014b], there is no need to explicitly add stuck-at faults at the cell boundaries to the candidate defect set, as all stuck-at faults are already included in the compact set as short defects between cell ports and power or ground.

Figure 3.19 shows per cell the simulation time in the defect characterization and test quality verification and compensation. Assuming we simulate the full set of short defects, for each library cell, the number of simulations equals the product of short-defect count and cell pattern count, resulting in 138,007,499 (=100%) defect-pattern tuples. With the approach proposed in this thesis, we first simulate 1,485,134 (=1.1%) defect-pattern tuples for the compact set. The simulation time based on the compact set is 96.8 hours. The 324 combinational cells take 10.8 hours, while the 152 sequential cells take 86.0 hours. Assuming per cell, all defect-pattern tuples take identical simulation time, the simulation time based on the full set would be 388.2 days, which is clearly unfeasible and underlines the necessity of reducing defect characterization time.

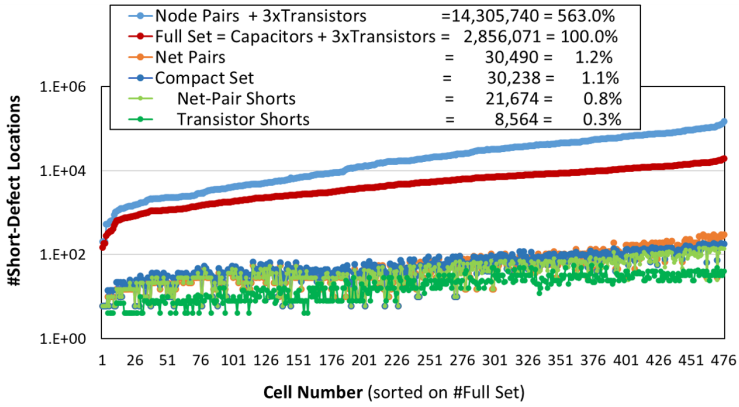


Figure 3.18: Full vs. compact sets of short-defect locations in GPDK045 library cells, and the division of the latter into net-pair and transistor shorts.

During short-defect characterization, we use the exhaustive set of one-cycle cell patterns to simulate each short defect and record the voltage and current values at each short-defect location for the downstream test quality verification and compensation. Using these values, we identify the combinations of net pairs and cell patterns that may result in different detection results due to different short-defect locations. In 2,168 (=10.0%) out of 21,674 net pairs, short defects are possibly non-equivalent for one or multiple cell patterns. Applying only the cell patterns that can cause different detections, we perform simulations for the other diagonal short defect of the 2,168 net pairs. The total additional number of simulated defect-pattern tuples is 24,200, and the corresponding simulation time is 1.4 hours. After this simulation, we determine 1,567 (=7.2%) net pairs between which two diagonal short defects are non-equivalent. Therefore, between these net pairs, not all short defects are equivalent. Among the 1,567 net pairs, 375 net pairs have only one cell pattern causing different detections for the two diagonal short defects. We already can determine the only two DDM columns caused by these net pair shorts without additional simulation. For the remaining 1,192 net pairs, we perform additional simulations for the full set of the short defects. 485,933 times simulations took additional 30.3 hours and identified 1,391 different DDM columns for these net pair shorts. In total, we spent 128.5 hours simulating the short defects; 75.3% simulation time is for the compact set of short defects, and 24.7% simulation time is for the test quality verification and compensation. Compared with the estimated simulation time for the full set, we save 98.6% time cost on defect simulation.

3.5 Conclusion

CAT promises to significantly reduce test-escape rates, as it targets cell-internal defects that are covered by conventional test approaches only on a serendipitous basis. Test quality and costs of CAT critically depend on the library characterization. To guarantee test quality, we propose a PEX-based approach to identify a full set of defect locations. To reduce the simulation time, we minimize the number of defects to be simulated without a negative impact on the test quality. We collapse the full set to a compact set based on the defect equivalence. For opens, this means that we consider only one defect per net branch, while for shorts, we consider only one defect per net pair. Even though simulation with the compact set maintains the test quality for a large number of defects, false positives can happen in some specific situations. On particular branches, open-defect location affects the resulting delay and may lead to different detection results based on the user-defined delay threshold. Between cell-internal net pairs, short-defect location affects the voltage, namely the logic value for the downstream circuit. We propose a solution to verify the test quality based on the compact set and, if necessary, restore the test quality to the same level based on the full set. In our experiment results for Cadence's GPDK045 library, the defect collapsing reduces the total number of defect locations to be simulated by 97.0% compared to the number of defects in the full set. To restore the test quality, we identify 1,391 more different DDM columns for the net pairs with non-equivalent short defects. In theory, we would reduce 97.3% simulation time compared with simulating the full set of defects.

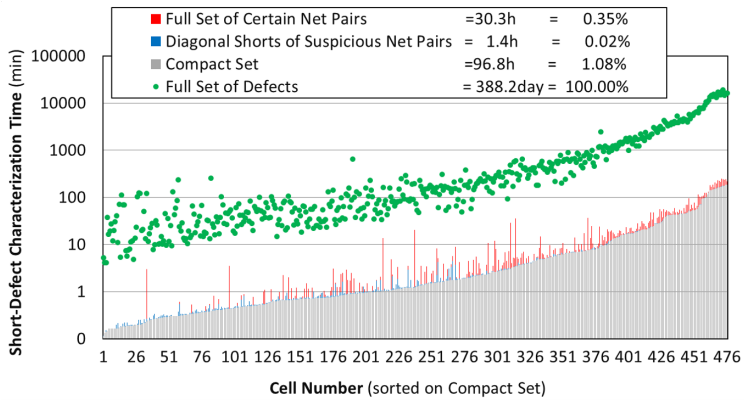


Figure 3.19: Simulation times for the short defects in GPDK045 library cells.

CHAPTER 4

Optimization of Cell-Aware ATPG Results by Manipulating DDMs

DDMs are used as the interface between the library characterization and cell-aware ATPG. In DDMs, the defects causing *static faults* are detectable by one-cycle cell patterns. This chapter proposes two algorithms that manipulate one-cycle cell patterns in DDMs to optimize the results of cell-aware ATPG for the static faults. The optimization effectiveness is quantified by the three ATPG KPIs in order of decreasing priority: fault coverage, test pattern count, and compute time. Algorithm 4.1 identifies don't-care bits in cell patterns, such that the ATPG tool can exploit the corresponding partly-specified cell patterns during cell-to-chip expansion to increase fault coverage and reduce the number of test patterns. Algorithm 4.2 selects, at cell level, a subset of *preferential* patterns that jointly provide maximal fault coverage at a minimized stimulus care-bit sum. To keep the ATPG compute time under control, we run cell-aware ATPG with the preferential patterns first, and a second ATPG run with the remaining patterns only if necessary. Selecting the preferential patterns maps onto a well-known $\mathcal{NP}$-hard problem, for which we derive an innovative heuristic that outperforms solutions in the literature. Figure 4.1 shows the position of the two proposed DDM manipulation algorithms in the CAT flow. The two algorithms can be used in isolation or conjunction. For twelve circuits, the

experimental results show that using them in conjunction delivers the best results of detecting the static faults.

The remainder of this chapter is organized as follows. We describe the motivation and detailed operation of Algorithm 4.1 and 4.2 in Sections 4.1 and 4.2, respectively. Section 4.3 provides experimental ATPG results and Section 4.4 concludes this chapter.

4.1 Don't-Care Bit Identification Algorithm

For some cell-internal faults, detection requires all inputs of the cell to be specified. However, many faults can be detected while one or more cell inputs are don't-care. Figure 4.2 shows an example of both cases in the AND2X1 cell from the Cadence GPDK045 library [Cadence Design Systems, 2014]. Locations of two cell-internal short defects are shown in Figure 4.2(a). Defect *d1* is a short between source and drain of NMOS transistor *N1* controlled by input *A* and *d2* a short between nets *n1* and *VSS*. To detect *d1*, both inputs should be specified. For *d2* detection, when either input is specified as '0', the other input can be don't-care. Figure 4.2(b) shows all fully- and partly-specified patterns for *d1* and *d2*. Defect *d1* is detected by only fully-specified pattern *p2*; defect *d2* can be detected by all fully- and partly-specified patterns except pattern *p4*.

We first introduce some notation conventions. Let *FP* be a set of fully-specified patterns and *PP* an extended pattern set including *FP* and all partly-specified patterns derived from *FP*. Function $r : PP \times [1, m] \longrightarrow \{\mathrm{L}, \mathrm{H}\}$ with $r(p, j) = s$ gives the response value s of pattern p on output bit j, where 'L' and 'H' denote logic 0 and 1, respectively.

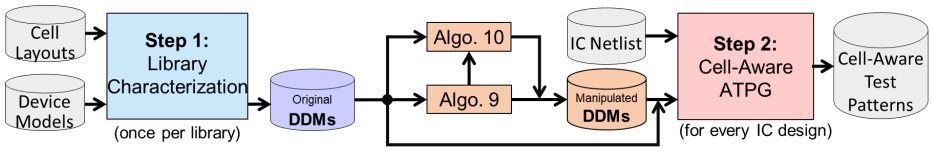
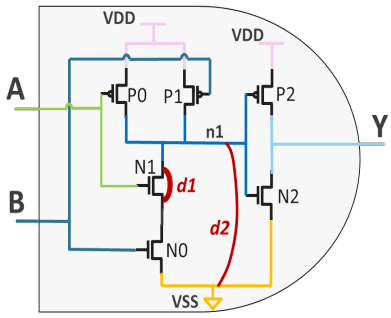


Figure 4.1: None, one, or both Algorithms 1 and 2 proposed in this paper can be used in the CAT tool flow to manipulate cell-patterns in DDMs to optimize the subsequent ATPG results.



(a) $d1$ and $d2$ in cell schematic.

Patterns		Defects	
	AB/Y	d1	d2
p1	00/L		1
p2	01/L	1	1
p3	10/L		1
p4	11/H		
p5	0X/L		1
p6	X0/L		1

(b) DDM for $d1$ and $d2$.

Figure 4.2: Examples of faults requiring either fully- or partly-specified cell patterns for detection in an AND2X1 cell.

Algorithm 4.1: DON'T-CARE BIT IDENTIFICATION ALGORITHM

01: **input**: F, FP, $DDM(F, FP)$;
02: **output**: PP, $DDM(F, PP)$;
03: *dictionary* = $\varnothing$; *PPdictionary* = $\varnothing$; // Initialization
04: // Step A: Divide fully-specified patterns in lists, based on response bits
05: **for all** $j \in [1, m]$ **do** {
06: *dictionary*[o_j,'L'] = $\varnothing$; *dictionary*[o_j,'H'] = $\varnothing$; // o_j is the output pin name
07: **for all** $p \in FP$ **do** {
08: append(*dictionary*[$o_j, r(p, i)$], p);
09: } }
10: // Step B: Identify all partly-specified patterns for each list of *dictionary*
11: **for all** $i \in [1, m]$ **do** {
12: Quine-McCluskey Algorithm identifying PP for *dictionary*[o_i,'L']
13: Quine-McCluskey Algorithm identifying PP for *dictionary*[o_i,'H']
14: // Record the corresponding fully-specified pattern set for pp
15: **for all** $pp \in PP$ **do** { *PPdictionary*[pp] = FP_{pp}; }
16: }
17: // Step C: Identify the detection set for each partly-specified pattern pp
18: **for all** $pp \in PP$ **do** {
19: **for all** $f \in F$ **do** {
20: $DDM(f, pp) = DDM(f, FP_{pp}[1]) \wedge ... \wedge DDM(f, FP_{pp}[|FP_{pp}|])$; } }

We develop Algorithm 4.1, that extends a given DDM containing fully-specified patterns with all partly-specified cell patterns that are implied by the original DDM. A pattern consists of stimulus bits for all n inputs and a response bit on exactly one of m outputs. To determine if two given patterns p_1 and p_2 can generate a pattern with one additional

don't-care stimulus bit, p_1 and p_2 should be identical in all bits, except for one stimulus bit j, where the two patterns should have opposite logic values (thus, $p_1(j)$ ='0' and $p_2(j)$ ='1' or vice versa). Algorithm 4.1 consists of three main steps, which are described below. Steps A and B are based on the well-known Quine-McCluskey algorithm [McCluskey, 1956].

Step A: We partition the fully-specified pattern set *FP* into at most $2m$ lists of which each has an identical response value on one of the m outputs. Variable *dictionary* stores these pattern lists (Lines 04–09). An example is shown as column "0 'X' Bits" in Figure 4.3.

Step B: For each pattern list, we identify all partly-specified patterns of *PP* that correspond to the fully-specified patterns in the list. Both *pp* $\in$*PP* and its corresponding fully-specified patterns FP_{pp} are recorded for the next step (Lines 10–16).

Each list identified in Step A is displayed in a matrix [McCluskey, 1956]; the columns in this matrix display the cell patterns with increasing 'X' count, while the rows correspond to increasing '1' count – see Figure 4.3. The fully-specified patterns end up in the column named *0 'X' Bits*. The green numbers in front of patterns are the identifiers of the patterns. We can add a pattern into Column $(i+1)$ *'X' Bits* on input j $(1 \leq j \leq n)$ if and only if in Column i *'X' Bits* we already have two patterns identical to the new partly-specified pattern, apart from input j where one pattern has a '0' and the other pattern has a '1'. Iteratively, we search the patterns in Column i *'X' Bits* and identify patterns for Column $i+1$ *'X' Bits*, starting of course with the given fully-specified patterns in Column *0 'X' Bits*. The algorithm will certainly end in Column n *'X' Bits*, but in typical cases sooner. As per column, we are searching for patterns which differ in only one stimulus bit, we can reduce compute time by considering only pattern pairs in adjacent rows in the same column. For instance, in Column *0 'X' Bits*, we consider every two patterns from Rows *1* and *2 '1' bits*, respectively and from Rows *2* and *3 '1' bits*,

	Cell AOI21: Response 'L' on Output 'Y'		
	0 'X' Bits	1 'X' Bit	2 'X' Bits
1 '1' Bit	1 001/L ✓	1 3 0X1/L ✓ 1 5 X01/L ✓	1 3 5 7 XX1/L
2 '1' Bits	3 011/L ✓ 5 101/L ✓ 6 110/L ✓	3 7 X11/L ✓ 5 7 1X1/L ✓ 6 7 11X/L	
3 '1' Bits	7 111/L ✓		

Figure 4.3: Example matrix used in Step B in Algorithm 1 (based on [McCluskey, 1956]).

respectively. 'X' cannot be identified from two patterns that are from the same row or from the non-adjacent rows *1* and *3 '1' bits*, respectively, because the difference of the bit '1' count between the two patterns indicates that at least two stimulus bits are different. In Figure 4.3, one pattern with two don't-care bits is displayed in Column *2 'X' Bits*: 'XX1/L'. As indicated in green, this pattern originates from four fully-specified patterns {1, 3, 5, 7} = {001/L, 011/L, 101/L, 111/L}. An important difference between the usage of the Quine-McCluskey in logic optimization [McCluskey, 1956] and in our application is that in the former, only patterns that we cannot derive more don't-care bits from are kept, shown as the two without check-mark in the example of Figure 4.3. Instead, in the latter we use *all* patterns (eleven in Figure 4.3) as each pattern is also associated with a detection set, and we need the information of both pattern and detection set for ATPG.

Step C: In this step, the detection sets for the new partly-specified patterns are calculated. A fault f is included in the detection set of partly-specified pattern *pp* if and only if f is element of the detection sets of *all* original fully-specified patterns that were used to derive *pp* (Lines 17–20).

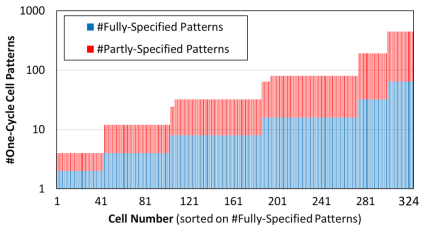
For 324 combinational cells of Cadence's GPDK045 45nm CMOS library [Cadence Design Systems, 2014], we identify 4,766 fully-specified cell patterns after 8 hours of analog simulations. From these one-cycle fully-specified cell patterns, Algorithm 4.1 derives 13,266 additional partly-specified cell patterns (= 2.8× increase), shown in Figure 4.4(a), in which 20,466 (= 30%) bits are identified as don't care. 281 (= 86.7%) out of 324 library cells have partly-specified cell patterns detecting at least one cell-internal fault, except 69 cells which are single-input cells (e.g. inverter and buffer) and XOR cells. In Figure 4.4(b), we compare the average number of care bits per fully-specified cell patterns, which is equal to the number of cell inputs, with the average number of care bits per pattern in the extended pattern set containing both fully- and partly-specified cell patterns. With the determined don't-care bits in the 281 extended pattern sets, the average number of care bits per pattern decreases. The minimum number of care bits among all partly-specified cell patterns is also shown in Figure 4.4(b). The largest difference between the numbers of cell inputs and the minimum care bits happens in AOI222, OAI222, AOI33, and OAI33 type of cells. Some defect detections require only two care bits compared with six care bits in all fully-specified cell patterns. Algorithm 4.1 is implemented by Python language, and the compute time is just ∼1 minute. However, this extended cell pattern set significantly increases ATPG compute time, which we reduce in the next section of this chapter by Algorithm 4.2.

4.2 MinCover Algorithms

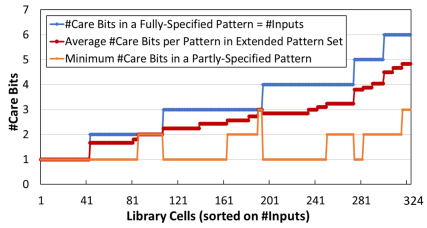
A library cell's DDM can contain many cell patterns, especially if it is extended to include all applicable partly-specified patterns. Typically, there is quite some overlap in the fault detection sets of any pair of cell patterns. Successful pattern expansion from cell-to-chip by the ATPG engine depends on the circuitry surrounding the cell in question, and is not guaranteed. Consequently, the order in which ATPG tries to expand the various cell patterns in a DDM has a big impact on the ATPG compute time. Ideally, we want to expand a small subset of the DDM patterns for which the union of their detection sets provides maximum fault coverage at cell level. However, a traditional ATPG engine has no insight in overlap of the various DDM patterns. Also, by expanding cell patterns with fewer specified bits, we increase the probability of successful expansion.

In this section, we describe Algorithm 4.2, which computes a minimized subset of DDM patterns with maximal cell-level fault coverage, i.e., identical to the combined fault coverage of all cell patterns in the DDM. We refer to the resulting patterns as the *preferential* patterns for that cell. We show that identifying a minimal set of preferential patterns for a cell is an $\mathcal{NP}$-hard problem, we present a framework that enables us to efficiently describe all prior algorithms addressing this problem, and subsequently present an innovative and effective heuristic algorithm that outperforms the prior art.

We use the subset of preferential cell patterns for two purposes. At library level (even before a specific chip design on the basis of this library is available), the size of the preferential pattern subset is a useful metric for how many patterns a particular library cell will need as a minimum. However, the main usage of the preferential pattern subset is during ATPG on a specific design, where we run ATPG in two stages. In Stage 1, we only try to expand the preferential patterns of the cell instances in the design. If



(a) The numbers of fully- and partly-specified two-cycle cell patterns.



(b) The numbers of care bits in one-cycle cell patterns.

Figure 4.4: Experimental results of Algorithm 4.1 on one-cycle fully-specified cell patterns.

the entire set of preferential cell patterns expands successfully, we have obtained maximal fault coverage with the smallest-possible number of cell patterns at minimal ATPG compute time. However, typically a small fraction of the preferential patterns fail to expand to chip level, and therefore the chip-level fault coverage is not maximal. Thus, in Stage 2, we run ATPG again, now providing the DDM information of the remaining (non-preferential) patterns for all cell instances. This two-stage ATPG keeps the ATPG compute time under control.

For the input DDM of the preferential pattern selection algorithm, it is not necessary to include the complete set of cell patterns given by the library characterization. The CAT tool users can specify a partial DDM out of the original DDM as the input to use the algorithm based on their own demands. We have some variable definitions for the preferential pattern selection problem. P is the full pattern set in a DDM. $P' \subseteq P$ represents the patterns in the input DDM which a tool user specifies. D is the set of all defects, for which the locations were identified as described in Chapter 3. $F \subseteq D$ is the set of all detectable defects, which we also call faults. $F' \subseteq F$ is the union of the detection sets for all patterns in P'.

Problem 1: Given fault set F and pattern sets P and P' with $P' \subseteq P$. Each $p \in P$ is associated with a detection set $ds(p) \subseteq F$ and $\bigcup_{i \in P'} ds(i) = F' \subseteq F$. Find a minimal set $MC \subseteq P'$ such that $\bigcup_{i \in MC} ds(i) = F'$.

Problem 1 is equivalent to the Set Covering Problem, which was proven to be $\mathcal{NP}$-hard [Karp, 1972]. Therefore, we developed a heuristic algorithm framework called MINCOVER to optimize MC. The algorithm framework is outlined in Algorithm 4.2. In every iteration, we employ a routine to move patterns from P' to either MC ('select') or *REST* ('deselect') (Line 05) and update P' and F' accordingly (Lines 06–07), until F' has been exhausted (Line 08).

Algorithm 4.2: MINCOVER Framework

```
01: input: F', P, P', DDM(F, P);
02: output: MC, REST;
03: MC = ∅; REST = ∅; // Initialization
04: repeat {
05:    Select/Deselect non-empty set of patterns Pat ⊆ P';
06:    if Select then {MC = MC ∪ Pat; P' = P'\Pat; F' = F' \ ⋃_{i∈Pat} ds(i);}
07:    else { REST = REST ∪ Pat; P' = P'\Pat; }
08: } until F' = ∅
```

The MINCOVER algorithm framework has numerous algorithm instances, which all differ with respect to the sequence of select/deselect routines which are employed in Line 05. We refer to the various routines we have developed with a single letter. A spe-

cific algorithm instance of MINCOVER is expressed by a regular expression describing a string of these letters, which represents the sequence of routines employed subsequently in Line 05. The various routines which we use are described below.

Definition 4.2.1 Pattern p is called an *essential* pattern if and only if $\exists_{f \in F'}\ ps(f) = \{p\}$.

To cover all faults, all essential patterns must be selected. Typically, essential patterns also detect other faults, for which they are not essential. As essential patterns need to be selected anyway, we better do this early in the algorithm, thereby avoid having to include other cell patterns covering the faults that are detected by essential patterns. The routine of selecting all essential patterns for a given DDM is denoted by letter 'E', has compute complexity $\mathcal{O}(|F'|)$, and is described as follows:

E: *Pat*= $\emptyset$; **for all** $f \in F'$ **do** { **if** $|ps(f)| = 1$ **then** *Pat*=*Pat*$\cup\ ps(f)$}.

Definition 4.2.2 Pattern p is called a *subset* pattern if and only if $\exists_{i \in P'}\ ds(p) \subseteq ds(i)$.

The ATPG engine does not need to expand subset patterns, provided their superset patterns can be expanded successfully. Hence, the subset patterns can be deselected and moved from P' to *REST*. Routine S, in which for a given DDM all subset patterns are collected in *Pat* consists of two nested loops and hence has compute complexity $\mathcal{O}(|P'|^2)$:

S: *Pat* $= \emptyset$; **for all** $p_1, p_2 \in P'$ **do** { **if** $p_1 \neq p_2 \wedge ds(p_1) \subseteq ds(p_2)$ **then** *Pat* = *Pat* $\cup\{p_1\}$}.

For many DDMs, routines E and S alone are insufficient to assign all DDM patterns to either MC or *REST*. In that case, we resort to a *greedy* selection of a single cell pattern which we add to the selected pattern set MC. We have formulated two greedy routines differentiated only by their cost functions $h(p)$. In both greedy routines, with compute complexity $\mathcal{O}(|P'|)$, a pattern p_g is selected for which holds $h(p_g) = MAX_{p \in P'}\ h(p)$.

tmp = 0;
for all $p \in P'$ **do** {**if** $h(p) \geq$*tmp* **then** *tmp*=$h(p)$; $p_g = p$};
Pat = $\{p_g\}$

In greedy routine G, we pick a pattern with the largest detection set, i.e. $h(p) = \sum_{f \in ds(p)}\ DDM(f,p) = |ds(p)|$. Our second greedy routine, denoted by letter W, assigns a *weight* $h(p) = \sum_{f \in ds(p)} \frac{DDM(f,p)}{|ps(f)|}$ to each pattern $p \in P'$. The weight $h(p)$ increases if p has a large detection set $ds(p)$, but also increases for detected faults $f \in ds(p)$ that have a small pattern set $ps(f)$. Faults with a small pattern set have only

few alternatives for detection, and hence such a pattern has a higher probability of being expanded anyway. Trying expansion of such a pattern early on might avoid to include unnecessary cell patterns. In prior work addressing the Set Covering Problem, a classic heuristic solution is the Greedy Algorithm [Johnson, 1974], denoted in our framework as G+: routine G is iteratively performed until F' is empty. Chvátal [Chvátal, 1979] showed that solutions obtained by G+ can be at most a factor $ln|F'|$ larger than an optimum solution. Selecting all essential patterns before applying the Greedy Algorithm (EG+) can reduce this factor from at most $ln|F'|$ to $log|F'|$, which was proven later by Qayyum [Qayyum et al., 2002]. Agathos & Papapetrou [Agathos and Papapetrou, 2013] used routines S, E, and G to compose two optimization algorithms, viz. SEG+ and (SE)+G+, in a communication application.

We make the following observations regarding algorithm compositions. Essential patterns should be selected first: E. Selecting essential patterns can introduce new subset patterns, and vice versa. Deselecting subset patterns (S) can introduce new essential patterns on the still unclassified remaining test patterns in the DDM. Applying sequences of only E and S routines can lead to a situation where no further essential or subset patterns can be identified, while still not all patterns in the DDM are classified as selected or deselected. In such a case, we might need to apply a greedy routine (G or W) to make progress. Execution of G or W cannot introduce new essential patterns, but it can generate new subset patterns. Performing S only before G or W does not make sense, because greedy selection will never select subset patterns and S has higher compute complexity.

We have defined a number of MINCOVER algorithms: seven using greedy routine G, and four using greedy routine W. Table 4.1 lists the aggregate results for these 11 MINCOVER algorithms on 548 library cells of Cadence's GPDK045 library [Cadence Design Systems, 2014]. The algorithms with our novel greedy routine W consistently outperform the algorithms based on greedy routine G, which is described in the literature. Based on the observations above, we expected algorithm composition (ES)+(W(SE)+)+ to perform the best, and the experimental results in Table 4.1 indeed confirm that.

For the total 4,766 one-cycle cell patterns, the best MINCOVER algorithm (ES)+(W(SE)+)+ identifies 1,399 (= 29.4%) preferential cell patterns covering all the detectable cell-internal defects. The number of selected essential and greedy patterns per cell in the GPDK045 library is shown in Figure 4.5. The number of exhaustive cell patterns is $m \times 2^n$, where n is the number of cell inputs and m is the number of outputs. The cells in Figure 4.5 are sorted on the number of cell inputs, and therefore the number of the exhaustive fully-specified cell patterns exponentially increases along x axis. However, the number of minCover cell patterns does not increase correspondingly, which means for complex cells, the required cell pattern count is much lower than the patterns generated by gate-exhaustive test. The most complex cell we used has six inputs and one output, and therefore the largest number of the exhaustive fully-specified

Algorithm Composition		Preferential One-Cycle Patterns		
		Essential	Greedy	Total
G+	[Johnson, 1974]	0	1429	1429
SEG+	[Agathos and Papapetrou, 2013]	1038	375	1413
EG+	[Qayyum et al., 2002]	1007	408	1415
(SE)+G+	[Agathos and Papapetrou, 2013]	1200	210	1410
(ES)+G+		1200	210	1410
((SE)+G)+		1340	68	1408
(ES)+(G(SE)+)+		1340	68	1408
EW+		1007	396	1403
(SE)+W+		1131	270	1401
(ES)+W+		1131	270	1401
(ES)+(W(SE)+)+		1190	209	1399

Table 4.1: Results for 11 MINCOVER algorithm compositions for original DDMs of 324 library cells in the GPDK045 library.

cell patterns is 64, while the minimum number of cell patterns covering all cell-internal defects is between [5, 10].

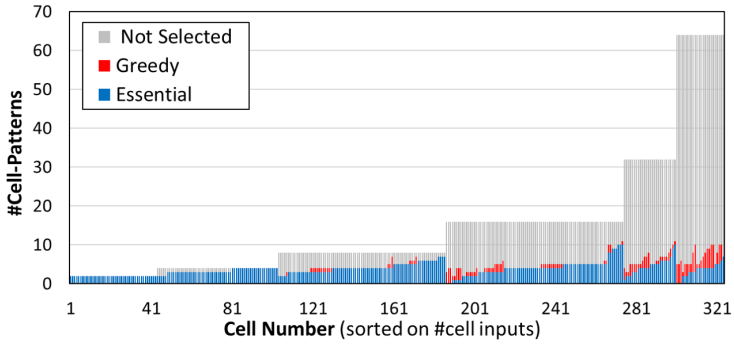


Figure 4.5: Results of algorithm (ES)+(W(SE)+)+ on 324 combinational cells of the GPDK045 library.

For a DDM containing only fully-specified cell patterns, the minimized number of preferential patterns leads to also the minimized sum of care bits in the these patterns. However, for a DDM containing partly-specified cell patterns, what we are after is not minimization of the number of preferential patterns, but minimization of the sum of care bits in the preferential patterns. Fewer care bits implies (1) lower probability for conflicts and hence more successful expansions, which should increase fault coverage and reduce test pattern count, (2) fewer specified bits in the chip-patterns, which should allow more compaction and reduce test pattern count, and (3) less work to expand them, which implies reduced ATPG compute times.

Problem 2: Given fault set F and pattern sets P and P' with $P' \subseteq P$. Each $p \in P$ is associated with a detection set $ds(p) \subseteq F$ and $\bigcup_{i \in P'} ds(i) = F'$. All $p \in P$ have n stimulus bits of which $cb(p)$ are care bits. Find a set $\boldsymbol{MB} \subseteq P'$ with minimal $\sum_{i \in MB} cb(i)$ such that $\bigcup_{i \in MB} ds(i) = F'$.

Problem 2 is $\mathcal{NP}$-hard; the proof is as follows.

Proof: Assume Problem 2 is *not* an $\mathcal{NP}$-hard problem. If for a certain problem instance $\forall_{p \in P'} cb(p) = n$, then $\sum_{i \in MB} cb(i) = n \times |\boldsymbol{MB}|$. In that case, Problem 2 becomes Problem 1, viz. find a minimum set $\boldsymbol{MB}$ such that $\bigcup_{i \in MB} ds(i) = F'$. Problem 1 was already shown to be $\mathcal{NP}$-hard. This implies a contradiction with our assumption and hence Problem 2 *is* $\mathcal{NP}$-hard.

The algorithm we propose for Problem 2 is based on what we have learned from the algorithms for Problem 1. Essential patterns should be selected first if we want to cover all faults: routine `E`. Partly-specified patterns are typically subset patterns of their fully-specified counterparts. Therefore, we will not use routine `S`, as that will deselect many partly-specified patterns and that does not help to reduce the number of care bits. Finally, we define a new greedy selection routine, which we call `D`. It is a modified version of routine `W`, as we have seen that `W` consistently outperformed `G`. To let the cost factor that governs greedy selection also include the number of don't-care bits in a particular pattern, we use $h(p) = \sum_{f \in ds(p)} \left(\frac{DDM(f,p)}{|ps(f)|}\right) \times (n - cb(p) + x)$. Here, $x \in \mathbb{N}^+$ is a means to balance the relative importance of both cost-factor components $\sum_{f \in ds(p)} \left(\frac{DDM(f,p)}{|ps(f)|}\right)$ and $n - cb(p)$.

We experimented on the 324 library cells of the GPDK045 library[Cadence Design Systems, 2014] with all MINCOVER algorithms from Table 4.1, where we replaced greedy routines `G` or `W` by routine `D`, while varying x. Based on the observations above, we expected algorithm `ED+` to perform best, and that was indeed the case, shown in Table 4.2. In Figure 4.6, we show the pattern counts and care-bit sums identified by `ED+` for $x \in [1..500]$. The minimum care-bit sum is obtained for $x = 33$. The exact pattern and care-bit counts are given in the last row of Table 4.2. The care-bit sum is lower than with all other MINCOVER algorithms based on greedy routines `G` and `W`, as now this has become an explicit selection criterium in `D`, while the total preferential pattern count has only increased marginally from 1399 to 1402. The compute time for each algorithm implemented in C on all 324 library cells was ∼0.1 minutes and thus negligible.

Figure 4.7 shows per cell the preferential pattern count selected by algorithm `ED+` with $x = 33$. Please note the y axis has a logarithmic scale. For cells with six inputs and one output, the selected pattern count is between [5, 11], while the total number of patterns in the extended DDMs can be [290, 414].

Algorithm Composition		Preferential Patterns			Bits in Pref. Patterns	
		Essential	Greedy	Total	Care	Don't-Care
G+	[Johnson, 1974]	0	1429	1429	5092	0
SEG+	[Agathos and Papapetrou, 2013]	1007	408	1415	4980	67
EG+	[Qayyum et al., 2002]	1038	375	1413	5028	9
(SE)+G+	[Agathos and Papapetrou, 2013]	1200	210	1410	5013	9
(ES)+G+		1200	210	1410	4956	66
((SE)+G)+		1340	68	1408	5002	9
(ES)+(G(SE)+)+		1340	68	1408	4946	65
EW+		1007	393	1400	4761	207
(ES)+W+		1147	253	1400	4902	66
(SE)+W+		1139	261	1400	4959	9
(ES)+(W(SE)+)+		1139	261	1400	4828	140
ED+	($x = 33$)	1007	395	1402	4747	233
(ES)+D+	($x = 33$)	1139	262	1401	4819	155
(SE)+D+	($x = 1$)	1147	253	1400	4956	12
(ES)+(D(SE)+)+	($x = 33$)	1139	262	1401	4819	157

Table 4.2: Results for 15 MINCOVER algorithm compositions for extended DDMs of 324 combinational cells in the GPDK045 library.

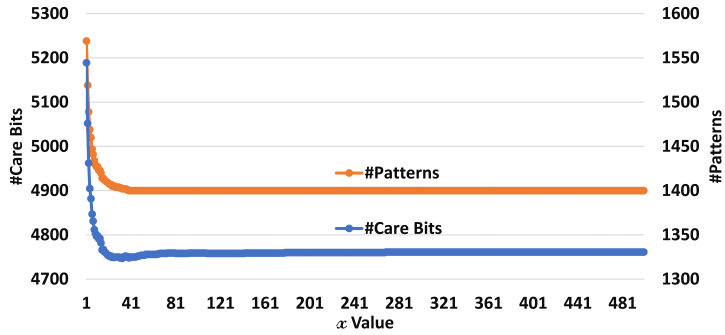


Figure 4.6: Results of algorithm ED+ for varing values of x on 324 combinational cells of the GPDK045 library.

4.3 Experimental Results

We performed cell-aware ATPG with Cadence's ATPG tool Modus (version 19.1) on twelve circuits [Santosh Malagi, 2019, Scott Davidson, 1999, OpenCores.org, 1999, Arm, 2007] that were first mapped onto Cadence's GPDK045 45nm CMOS library [Cadence Design Systems, 2014]. For each circuit, we compare the cell-aware ATPG results for various types of optimized DDMs versus the original DDMs generated by defect simulation. Let DDM_0 be the set of *original DDMs* of which cells are used in a given

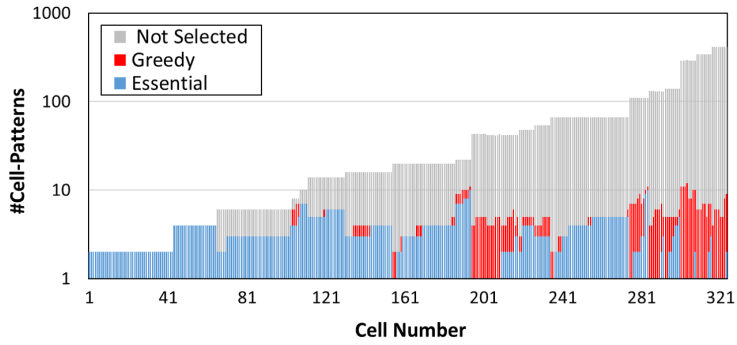


Figure 4.7: Results of algorithm ED+ with $x = 33$ on 324 combinational cells of the GPDK045 library.

	One-Stage ATPG	Two-Stage ATPG
Only fully-specified cell patterns	$\boldsymbol{DDM}_0$ (original DDMs, baseline)	$\boldsymbol{DDM}_1$ (i.e. $\boldsymbol{DDM}_0$ split based on preferential and remaining patterns)
Fully- and partly-specified cell patterns	$\boldsymbol{DDM}_2$	$\boldsymbol{DDM}_3$ (i.e. $\boldsymbol{DDM}_2$ split based on preferential and remaining patterns)

Table 4.3: Four types of DDM derived by Algorithms 4.1 and 4.2.

circuit gate-level netlist. The three types of *optimized DDMs*, named DDM_1, DDM_2, and DDM_3, were obtained by applying either or both algorithms proposed in this chapter, shown in Table 4.3. Partly-specified cell patterns are the outputs of Algorithm 4.1, i.e. don'-care bit identification algorithm. Therefore, the two types of DDM in the second row, DDM_2 and DDM_3, include don't-care bits. Algorithm 4.2 selects preferential cell patterns and requires to perform two-stage ATPG. DDM_1 and DDM_3 in the second column include two parts: the partial DDM with the preferential cell patterns and the remaining part. To minimize the care-bit sum in the preferential cell patterns, we apply MINCOVER algorithm (ES)+(W(SE)+)+ on DDM_0 to obtain DDM_1, and apply MINCOVER algorithm ED+ on DDM_2 to obtain DDM_3. The identified *preferential* cell patterns are prioritized and fed into the ATPG engine first, and the remaining cell patterns are fed subsequently if necessary. Based on the four types of DDM, we perform four ATPG experiments.

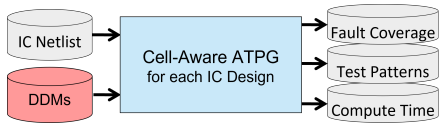
The one- and two-stage ATPG flows are shown in Figure 4.8. A successful cell-pattern expansion for the target cell results in logic values at inputs and outputs of the other cell instances in the expansion path. These logic values also detect some faults in the cor-

responding cell instances. Fault simulation interwoven with the cell-pattern expansions identifies additionally covered faults. A MINCOVER cell pattern expansion can cause some non-MINCOVER cell patterns for other cell instances in the expansion path. To obtain a correct fault coverage of each successful pattern expansion, we should offer the full information of DDMs for the first stage ATPG, and similarly, also for the second stage ATPG, shown as Figure 4.8(b).

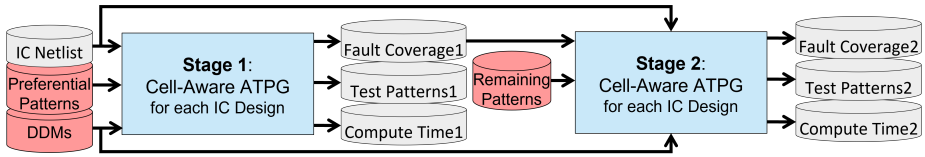
For the used twelve circuits, Table 4.4 shows the number of cell patterns before and after applying the two algorithms and Table 4.5 shows the number of care bits in the corresponding cell patterns.

The first main column of Table 4.4 gives the design data, more cell instances with more inputs per cell improve the effectiveness of our optimization algorithms. The GPDK045 library that we used in the experiments reported in this chapter is rather small and its cells have at most six inputs; this means that the benefits obtained with circuits based on GPDK045 are rather pessimistic with what would be possible for larger, industrial standard-cell libraries. The column 'Design Data' consists of three sub-columns: the circuit name, the number of combinational cell instances in the circuit $|I|$, and the weighted average number of inputs per cell instance $\sum_{k\in[1..6]} \left(k \times |\{i \in I | n_i = k\}|\right)/|I|$. The higher this weighted input count, the more effective our two algorithms are for the consistent design.

Subsequently, column 'number of cell patterns' presents the results of our DDM manipulation algorithms. The first and second sub-columns give the sum of all and the preferential fully-specified cell patterns in DDM_0 for all instances $i \in I$, while the third



(a) One-stage ATPG.



(b) Two-stage ATPG.

Figure 4.8: One- and two-stage ATPG flows based on DDMs non-optimized and optimized MINCOVER algorithm, respectively.

Design Data			Number of Cell Patterns						
Name	$\|I\|$	WAvg. Inp.	DDM_0	$Pref._0$	$Pref._0/DDM_0$	DDM_2	DDM_2/DDM_0	$Pref._2$	$Pref._2/DDM_2$
b15	2933	2.95	31800	12505	39.3%	92166	2.90×	12505	13.6%
b20	3212	2.65	30200	14985	49.6%	72682	2.41×	14985	20.6%
aes	3320	2.82	35284	13606	38.6%	111762	3.17×	13606	12.2%
b21	3482	2.63	32068	16520	51.5%	75666	2.36×	16520	21.8%
dtmf	3726	2.72	38542	15150	39.3%	121242	3.15×	15150	12.5%
b22	5036	2.64	46752	23668	50.6%	110196	2.36×	23668	21.5%
M0	5289	2.94	61568	20475	33.3%	207910	3.38×	20475	9.8%
b17	8981	2.94	97784	38043	38.9%	288230	2.95×	38043	13.2%
fpu	19136	2.97	220544	84950	38.5%	616994	2.80×	84950	13.8%
b18	20545	2.94	227466	88879	39.1%	659616	2.90×	88879	13.5%
M3	32626	2.95	388412	129329	33.3%	1293250	3.33×	129329	10.0%
b19	40298	2.93	441896	173509	39.3%	1283378	2.90×	173509	13.5%
Avg.	12382	2.84	137693	52635	40.9%	411091	2.88×	52635	14.7%

Table 4.4: The results of Algorithms 4.1 and 4.2 on the cell pattern count for twelve circuits. Circuit dtmf is from Cadence [Santosh Malagi, 2019]. Aes and fpu are from OpenCores [OpenCores.org, 1999]. M0 and M3 are from Arm [Arm, 2007]. All the other circuits are from ITC99 benchmarks [Scott Davidson, 1999].

Design Data			Number of Care Bits						
Name	$\|I\|$	WAvg. Inp.	DDM_0	$Pref._0$	$Pref._0/DDM_0$	DDM_2	DDM_2/DDM_0	$Pref._2$	$Pref._2/DDM_2$
b15	2933	2.95	122140	39713	32.5%	309736	2.54×	38468	12.4%
b20	3212	2.65	103676	43297	41.8%	229320	2.21×	42488	18.5%
aes	3320	2.82	143812	42130	29.3%	415573	2.89×	40869	9.8%
b21	3482	2.63	107832	46659	43.3%	234869	2.18×	45909	19.5%
dtmf	3726	2.72	151670	46165	30.4%	430595	2.84×	45190	10.5%
b22	5036	2.64	158148	67714	42.8%	341442	2.16×	66579	19.5%
M0	5289	2.94	253516	67252	26.5%	738554	2.91×	65407	8.9%
b17	8981	2.94	378564	121153	32.0%	976797	2.58×	117426	12.0%
fpu	19136	2.97	844492	271013	32.1%	2092037	2.48×	265413	12.7%
b18	20545	2.94	877958	282518	32.2%	2238668	2.55×	274329	12.3%
M3	32626	2.95	1602836	425207	26.5%	4615396	2.88×	414499	9.0%
b19	40298	2.93	1702888	549666	32.3%	4349339	2.55×	533648	12.3%
Avg.	12382	2.84	537294	166874	33.5%	1414361	2.56×	162519	13.1%

Table 4.5: The results of Algorithms 4.1 and 4.2 on the care-bit count for twelve circuits.

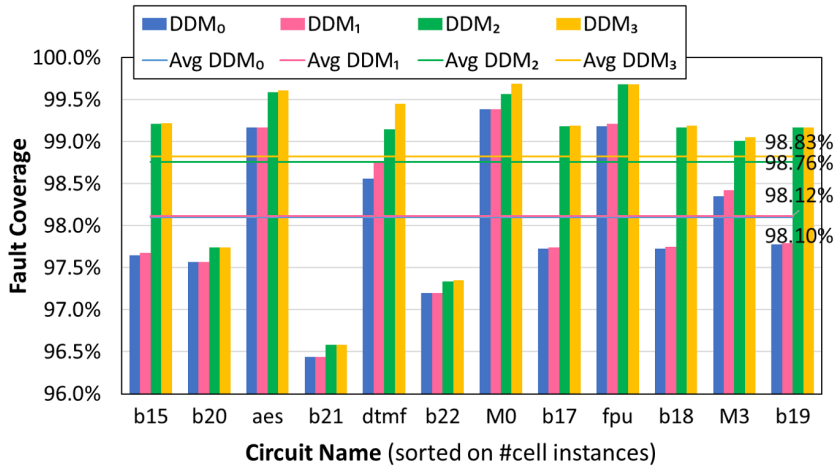
sub-column gives the percentage of the preferential patterns in DDM_0. The sum of both partly- and fully-specified cell patterns in the extended DDMs DDM_2 is shown in the fourth sub-column, while the fifth sub-column gives the increase from DDM_0 to DDM_2. Sub-column six list how many preferential patterns in DDM_2 are selected and the last sub-column expresses that number as a percentage of the pattern count in DDM_2. The smaller that percentage, the more complex cell instances are in the corresponding design. Therefore the percentage is inversely proportional to the weighted average number of inputs per cell instance.

Similarly, Table 4.5 shows the summed number of care bits in the cell patterns manipulated by the two algorithms.

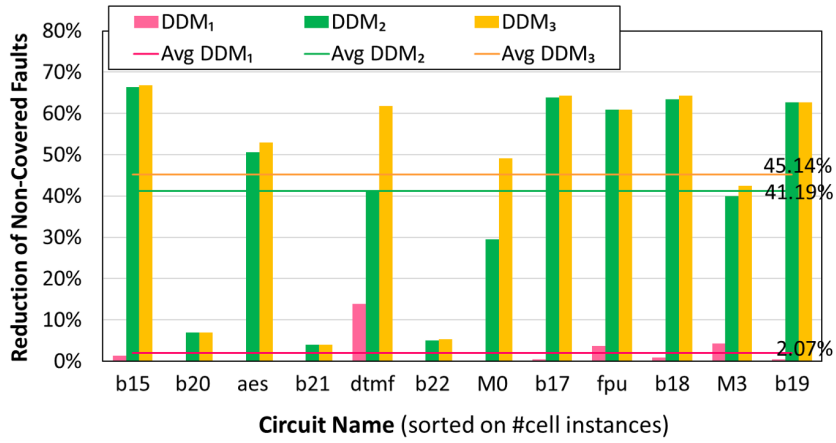
Figures 4.9 and 4.10 shows the optimization results of fault coverage, test pattern count, and compute time, which are sorted in order of decreasing priority. For each circuit in each sub-figure, four bars present the results of the four ATPG experiments based on the four types of DDMs, respectively. Figure 4.9(a) and Figure 4.9(b) express the fault coverage respectively the reduction of the non-covered fault fraction, i.e., 100% - fault coverage. For example: for benchmark circuit $b15$, the cell-aware fault coverage based on DDM_0 was 97.65%, while the fault coverage based on DDM_3 increased to 99.22%; this constitutes a reduction of the fraction of non-covered faults with $1 - ((1 - 99.22\%)/(1 - 97.65\%)) = 66.81\%$. The results on fault coverage show that (1) fault coverages based on the extended DDMs, DDM_2 and DDM_3, are always higher than the coverages based on DDM_0 and DDM_1 which only contain fully-specified cell patterns, and (2) based on identical DDMs, fault coverages based on the two-stage ATPG which expands the preferential cell patterns first are always larger than that by the one-stage ATPG. Partly-specified cell patterns means fewer expansion bits for the ATPG engine, so the probability of successful expansion is higher than the fully-specified cell patterns. If none of the fully-specified cell patterns of a cell-internal defect is expanded successfully, this defect may be covered by one of its partly-specified cell patterns. Therefore, the fault coverages based on DDM_2 and DDM_3 are higher than that based on DDM_0 and DDM_1. During ATPG, the cell-pattern expansion, fault simulation, and chip-level test pattern compression are interwoven. The two-stage ATPG cause the chip-level pattern compression is processed in two stages, and hence not effective, but the generated more test patterns leads to a slightly higher fault coverage. Compared to the ATPG based on the original DDMs DDM_0, the ATPGs based on DDM_1,DDM_2, and DDM_3 reduce the non-covered faults by 2%, 41%, and 45%, receptively. Using the two algorithms in conjunction gives the best test quality.

In Figure 4.10(a), we give the numbers of generated test patterns. DDM_2 and DDM_3 also bring benefits for the number of test patterns (indicating test execution time), compared with DDM_0 and DDM_1. On average, the DDM_2 and DDM_3 based ATPGs reduce the test pattern count by 14% and 10%, respectively. As mentioned above, the two-stage ATPG compared with the one-stage ATPG, brings benefit to test quality with slightly increased test patterns. The DDM_1 based ATPG has 2.7% more test patterns than the DDM_0 based ATPG, and the DDM_3 based ATPG gives 4.4% has test patterns than the DDM_2 based ATPG. Don't-care bits in DDM_2 and DDM_3 (on average for the twelve circuits 2.84$\times$ more cell patterns) bring benefits to fault coverage and pattern count. However, the ATPG execution time based on DDM_2 and DDM_3 are bound to increase on average 211.8% and 103.6%, respectively. The ATPG compute time comparison between DDM_2 and DDM_3 shows that the two-stage ATPG approach makes that the increase of ATPG compute time is reduced by 108.2%.

Figure 4.11 shows a summary of the optimization on fault coverage and test pattern



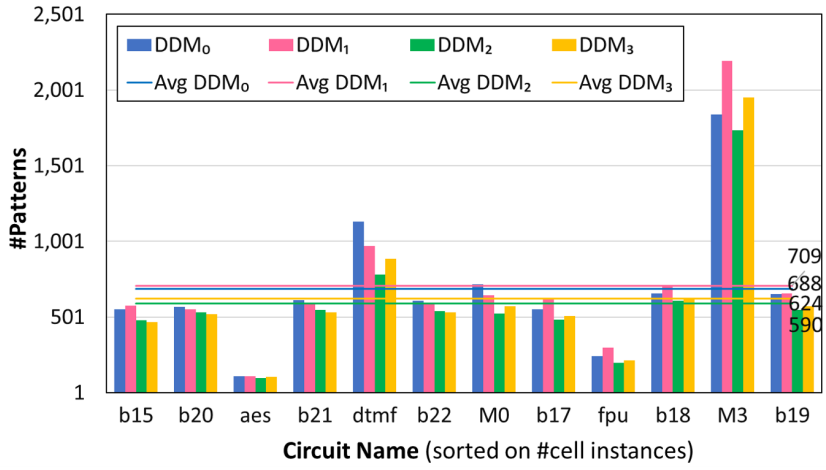
(a) Cell-aware fault coverage.



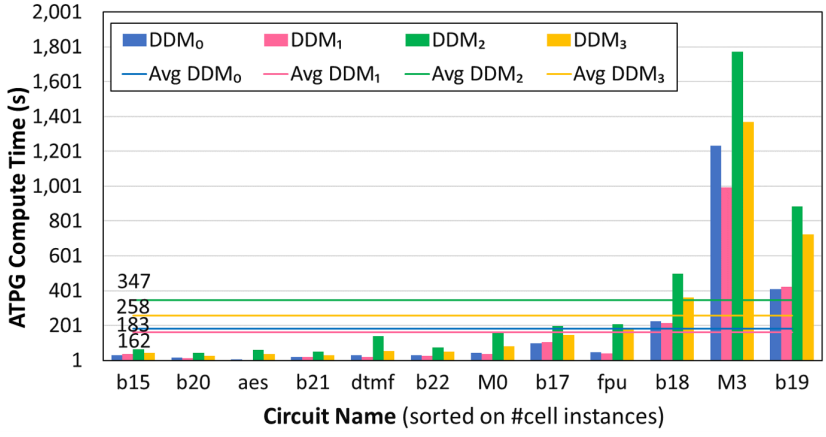
(b) Missed fault coverage reduction.

Figure 4.9: Fault coverage comparison.

count based on the four types of DDM. We use the average reduction percentages for the twelve circuits to draw four points in the figure. DDM_0, as the baseline, has optimization on neither fault coverage nor test pattern count. Compared to the results based on DDM_0, DDM_1 has a higher fault coverage, but costs more test patterns. DDM_2 and DDM_3 bring benefits on both fault coverage (representative for test quality) and test pat-



(a) Chip-level test pattern count.



(b) ATPG compute time.

Figure 4.10: Comparison of test pattern count and ATPG compute time.

tern count (representative for test execution time), but increase the ATPG compute time (representative for test development time). Considering the decreasing priority among test quality, test execution time, and test development time, the best ATPG results are achieved by using the two algorithms in conjunction, i.e. using DDM_3.

Figure 4.12 shows the weighted average number of inputs of the cell instances in a circuit

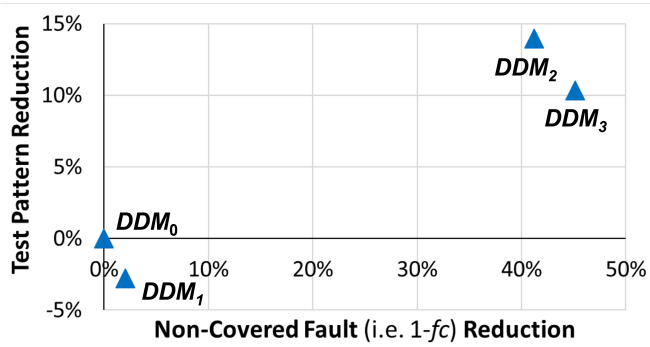


Figure 4.11: Optimization result comparison between the four types of DDM.

is a good indicator for the reductions in non-covered faults and test patterns that can be achieved by the combination of Algorithms 4.1 and 4.2 as described in this chapter. Only circuit $M3$ is an anomaly, perhaps due to its specific circuit structure.

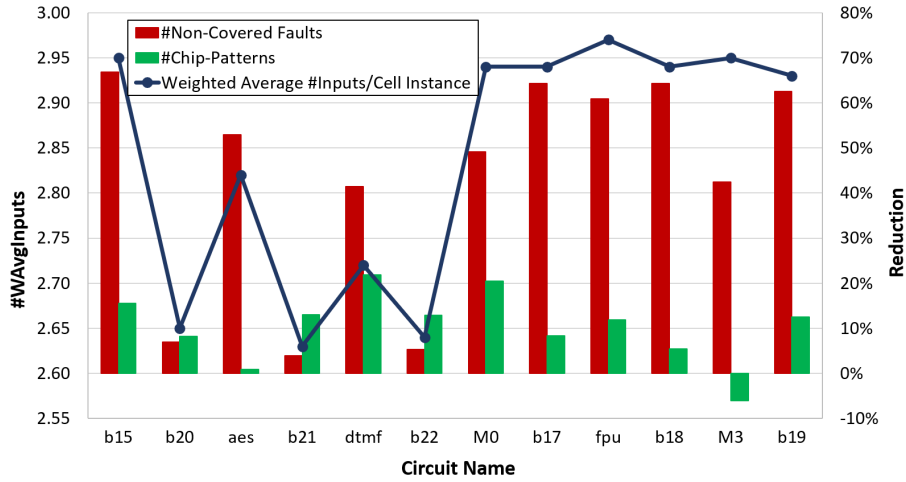


Figure 4.12: Optimization of ATPG results with minBit pattern sets.

4.4 Conclusion

DDMs represent the result of which cell-internal defects are detected by which cell patterns. This chapter presented two algorithms that manipulate one-cycle DDM patterns to improve the results of subsequent cell-aware ATPG runs for static faults. Algorithm 4.1 extends a given DDM to include all patterns with don't-care bits, to increase fault coverage and reduce test-pattern count. This typically leads to a large increase in cell patterns. Algorithm 4.2, an innovative heuristic solution for a well-known $\mathcal{NP}$-hard problem, identifies a subset of cell patterns with minimized care-bit sum, yet maximal fault coverage. These preferential patterns are tried for cell-to-chip expansion first by the cell-aware ATPG tool, while, if necessary, the remaining patterns are handled in a second ATPG run.

An attractive benefit of both algorithms is that they are executed as part of library characterization: once per library, instead of once per IC design. The two algorithms can be used either stand-alone and in conjunction. In this chapter, we discuss the results on both usages and show that using the algorithms in conjunction achieves the best results. For a set of twelve benchmark circuits, on average the non-covered faults were reduced with 45% and the test pattern count with 10%. The two-stage ATPG approach helped to keep the compute time under control. Despite that, the compute time increased, but as compute time is an non-recurring engineering cost only, we consider it acceptable in view of the improvements in the other two key ATPG performance indicators.

CHAPTER 5

Speeding up Library Characterization by Preceding Simulation with Structural Analysis

Library characterization is performed only once per library, but in today's implementations [Hapke et al., 2014b, Gao et al., 2019a] it is nevertheless a very time-consuming task. To generate a DDM for cell c, analog simulations are performed on the layout-extracted transistor-level netlist for cell c for all elements of the set of potential cell-internal defects D_c and cell patterns P_c. D_c typically contains several dozen of defect locations. P_c usually includes both one-cycle patterns to detect static faults and two-cycle single-transition patterns to detect delay faults. For a cell with k inputs and m outputs, this includes up to $k \times 2^k \times m$ two-cycle patterns and $2^k \times m$ one-cycle patterns. As the total number of simulations to run is large, characterization is a time-consuming operation; characterizing an entire library can easily take several weeks. For example: characterization of the Cadence GPDK045 library [Cadence Design Systems, 2014], with 476 cells, for the exhaustive set of one-cycle static test patterns and two-cycle single-transition patterns, takes approximately 83 days compute time.

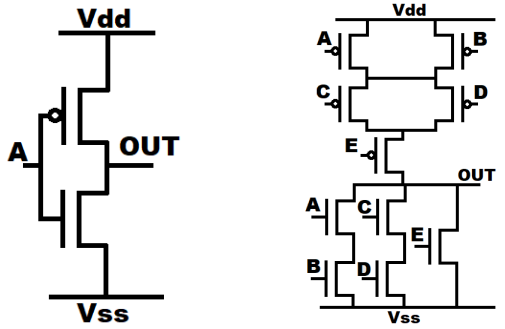
In this chapter, we propose a technique to effectively reduce the compute time required for CAT library characterization. A majority of the DDM entries is undetectable. Un-

detectability is not necessarily bad news: it just means that even if defect d is present, it will under cell pattern p not cause a behavioural anomaly at the cell's output. In this chapter, we present one reason why a (d, p)-tuple can be undetectable, which we refer to as *logical undetectability*. It can efficiently be checked on the basis of only the cell's topology as expressed in its transistor-level netlist. This chapter presents our *Undetectability Checker Algorithm* (UCA) that automatically performs a structural analysis of a cell, for every (d, p) tuple, to identify if it is logically undetectable. The simulation of the logically undetectable (d, p) tuples can be avoided, reducing the characterization time.

The remainder of this chapter is organized as follows. In Section 5.1 we present a model for library cells that is convenient for the structural analysis. Sections 5.2 and 5.3 describe respectively the structural analysis and the pseudo-code of DCA algorithm. Experimental results are presented in Section 5.4 and Section 5.5 concludes this paper.

5.1 Modeling of Library Cell Structure

To perform a structural analysis of the CMOS library cells, it is first necessary to define a model that is easy to handle during the structural analysis and that can describe all considered library cells. The cells considered in this work are 476 combinational and sequential logic cells from GPDK045 library. The physical-only cells, such as filler cells, decap cells or isolation cells were not considered, since they do not implement any logic function.



(a) INV (b) AOI221

Figure 5.1: Two CMOS combinational cells' schematics. Both cells have one compartment.

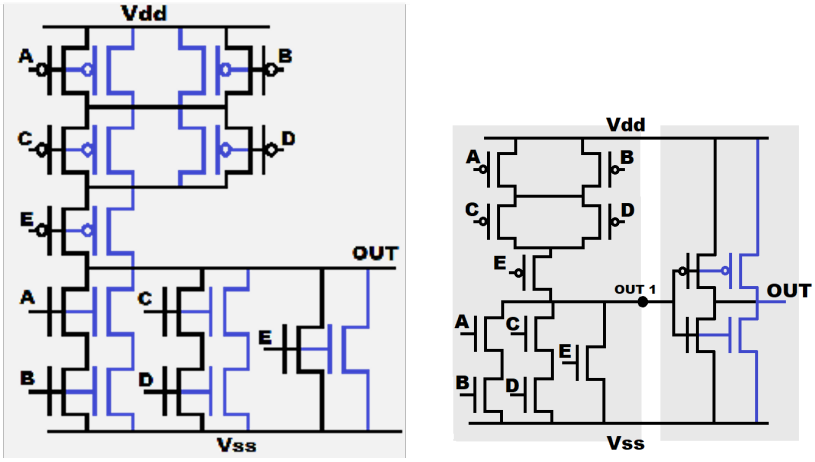
A fundamental concept for this analysis is the *compartment*. With compartment we mean

a structure that has an output net, that is connected to V_{DD} through a pull-up network and connected to V_{SS} through a pull-down network. The pull-up network is only composed of pMOS transistors and the pull-down network is only composed of nMOS transistors. A compartment includes transistors that have their gates controlled by external nets and includes nets that control external transistors (transistors of other compartments), but never includes nets that control internal transistors. Each compartment is logically and functionally independent and implements its own logic function. Figure 5.1 shows two simple CMOS cells composed of a single compartment. The INV cell in Figure 5.1(a) has one input *A*, and the AOI221 cell in Figure 5.1(b) has five inputs *A*–*E*. All transistors in both cells are controlled only by the inputs which request external signals.

Other CMOS cells can include multiple compartments. For example a AND cell is composed by a NAND compartment and by an INV compartment connected in sequence. The reason why these cells are designed is to optimize their layout, since from the logic point of view a *two-compartment* cell can be simply implemented using two independent cells. The proposed UC algorithm uses the concept of compartment to analyze how a defect that affects a compartment's output can propagate its faulty effect to the cell's main output, as it would do when propagating through multiple cells.

The cells shown in Figure 5.1 only contain the transistors necessary to implement the cell's logic function, we call them functional transistors. Some cells are designed to drive a high capacitive load and hence they include drive transistors in parallel to the functional ones. In this case the number of paths connecting the output to V_{DD}/V_{SS} does not change, but every path consists of multiple transistors that are activated by the same input signal. Cells with a high drive strength are less vulnerable to cell-internal defects, since the redundancy in the circuit makes them less dependent on a single branch. The experimental results collected from the GPDK045 library show that the number of defects detected in cells with high drive strength is lower than in the other ones and that only the hardest defects can be detected. This does not necessarily mean that the used testing technique is not good enough to detect them, but it means that the cell design is solid against potential defects. Figure 5.2 shows two cells with double drive strength (indicated by the *'X2'* in their names). Depending on the cell, the drive transistors can be placed in every compartment or only in the last one, since the last compartment is the one that will drive the capacitive load at the output. The *one-compartment* cell in Figure 5.2(a) directly drives the output net, hence it includes a drive transistor for every functional transistor. Differently, the *two-compartment* cell in Figure 5.2(b) only needs the additional transistors in the INV compartment driving the output. In some cases if the drive strength is very high and many drive transistors are required in the last compartment, also the previous compartment will have drive transistors to balance the high capacitance.

In this chapter we analyze Cadence's GPDK045 library. Out of 563 cells we excluded



(a) AOI221X2. (b) AO221X2.

Figure 5.2: Two CMOS cells with double drive strength. Gray boxes indicate compartments. (a) has one compartment. (b) has two compartments.

the physical only cells, such as decap or isolation cells, and ran our algorithm on the remaining 476 logic cells. Results collected from our structural analysis show that the most complicated cells can have up to 50 transistors, 6 inputs, and 7 compartments.

5.2 Defect Detectability Analysis

CAT characterization requires simulating for every library cell every potential short and open defect with every cell pattern. Given a cell c and a potential defect d, depending on the applied cell pattern p we can distinguish three different cases, shown in Figure 5.3. (1) (d, p) is called *logically undetectable* if d does not create a path that connects V_{DD} to V_{SS} in case d is a short defect or if cell pattern p isolates defect d in an inactive part of the cell in case d is an open defect. (2) (d, p) is called *electrically undetectable* if defect d is on an active path or connects V_{DD} to V_{SS}, but the resistance values of the cell paths are such that even the hardest version of d is undetectable. (3) (d, p) is *detectable* if defect d has an effect on the cell's output. Figure 5.4 shows a two-input AND cell with an internal short defect and all four cell patterns. Depending on which pattern is applied, the active paths inside the cell (represented in green) differ, consequently changing the effect of the defect on the cell output behavior.

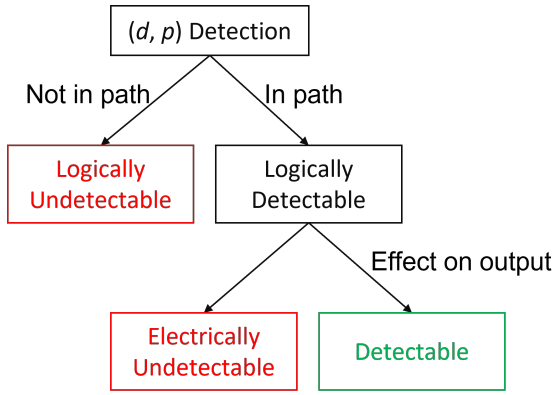
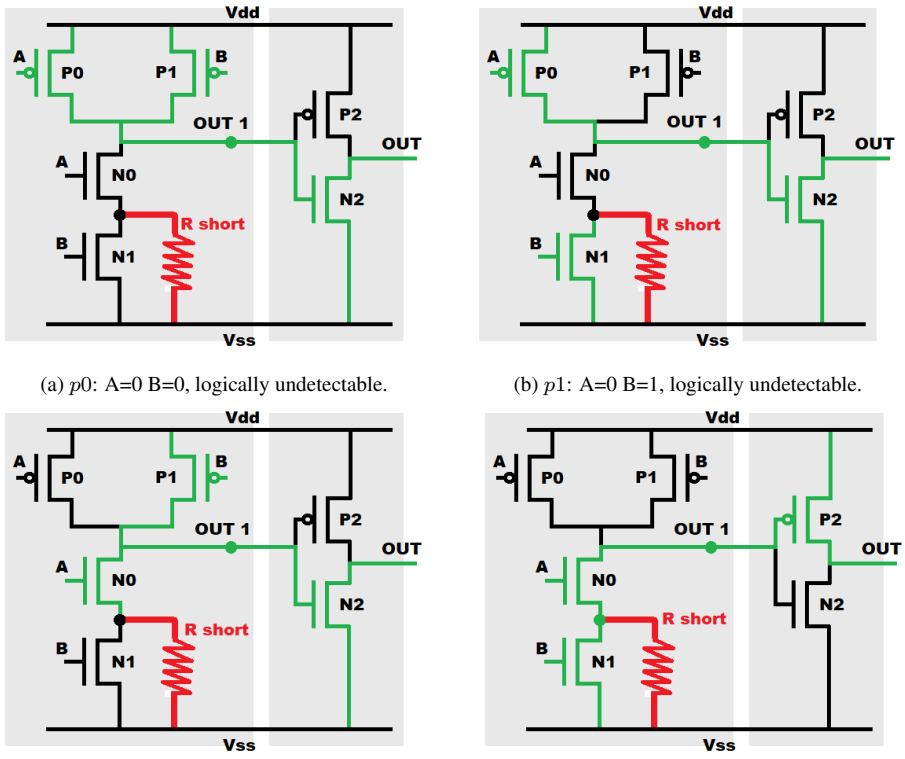


Figure 5.3: Detection results of tuple (d, p). If d is a short defect, 'path' should connect V_{DD} to V_{SS} due to d and p. If d is an open defect, 'path' is a set of nets activated by pattern p.

When a CMOS cell is operating normally, each net can be connected either to V_{DD} or to V_{SS}, otherwise the cell can be slowed down or a logic error can occur. Considering the case in Figure 5.4, in order to have an effect on the cell's output (OUT), the defect must affect the output of the first compartment (OUT 1). This can only happen if V_{DD} is connected to V_{SS} through a path generated by the defect. In the three cases in Figures 5.4(a), 5.4(b) and 5.4(d), the pattern cannot detect the defect. These (d, p) tuples are logically undetectable. In the other case in Figure 5.4(c), pattern p can potentially detect defect d, depending on the resistance values of the paths in the cell and of the defect, hence we call this defect-pattern tuple logically detectable. In the considered case the correct logic value of the output OUT is 0. Since the second compartment is controlled by the output of the first compartment, the logic value of OUT 1 must be 1. To achieve this, it must be: $V_{OUT1} > V_{TH}$, where V_{TH} is the high logic threshold. Assuming the source-drain resistances of the transistors $N0$ and $P1$ are R_{N0} and R_{P1}, and assuming that the parasitic resistances of the nets are negligible, the logic error occurs only if the following condition is satisfied:

$$R_{short} < R_{P1} \cdot \frac{V_{TH} - V_{SS}}{V_{DD} - V_{TH}} - R_{N0} \tag{5.1}$$

From this simple analysis it can be concluded that for a cell with a given cell pattern p and a defect d, if the tuple (d, p) is logically detectable the detectability depends on the resistance values of the cell and of the defect. In case the condition in Equation 5.1



(a) $p0$: A=0 B=0, logically undetectable.

(b) $p1$: A=0 B=1, logically undetectable.

(c) $p2$: A=1 B=0, logically detectable.

(d) $p3$: A=1 B=1, logically undetectable.

Figure 5.4: AND cell with short defect (in red) and all the cell patterns.

is met, the defect-pattern tuple is called detectable. Otherwise the tuple (d, p) is called electrically undetectable, since the undetectability of the defect-pattern tuple depends on the cell's electrical properties and on the defect's resistance value.

In case a defect is too weak to cause a logic error on the cell's output it is not detectable using a one-cycle pattern, but it may still be detected using a two-cycle pattern. In fact even if the defect cannot change the logic value of the cell's output, it can degrade the cell's dynamic properties and cause an excessive delay. To detect a defect using a two-cycle pattern, it is necessary to initialize the output to a specific logic value and then change it, activating a path that includes the defect. In this way, if the short defect has an effect on the cell's delay, it can be detected. Let's call the gate delay of the fault-free cell, when a specific two-cycle pattern is applied D_{nom}, and the gate delay with the

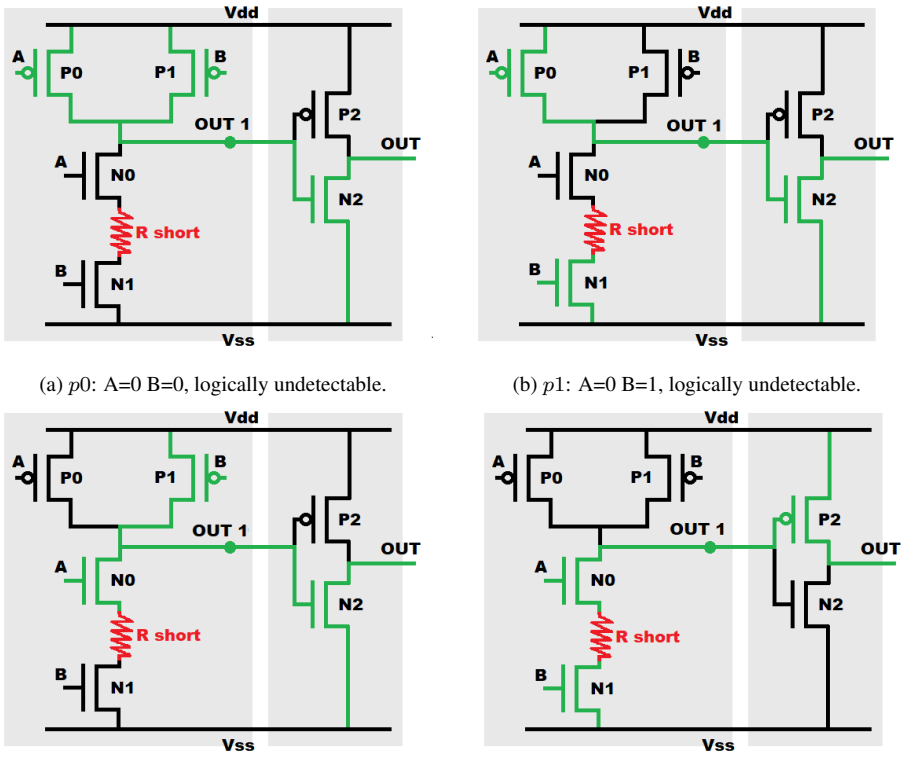
injected defect D_{fault}. If the delta delay $\Delta D = \frac{D_{fault} - D_{nom}}{D_{nom}}$ is higher than a pre-defined threshold value, the defect is considered detected. For a specific two-cycle pattern, if the resistance of the defect is hard enough to make the ΔD higher than the threshold value it is detectable, otherwise it is considered electrically undetectable. All the cell patterns $p \in P$ in logically undetectable (d, p) tuples are not useful to detect delay faults if they are in the second cycle of a two-cycle pattern. Referring to the example in Figure 5.4 the patterns p_0: A=0 B=0, p_1: A=0 B=1, p_3: A=1 B=1 cannot detect the defect. Consequently the two-cycle patterns that include one of these in the second cycle cannot detect the defect and their simulation is unnecessary.

As it has been done with short defects we can distinguish logically undetectable, electrically undetectable and detectable (d, p) tuples also for open defects. In fact if an open defect d is not part of an active path of the cell when pattern p is applied, it cannot be detected. Observing the example in Figure 5.5, the cell patterns shown in Figures 5.5(a), 5.5(b) and 5.5(c) do not activate the defective part of the cell, so these (d, p) tuples are logically undetectable. In the example shown in Figure 5.5 only the two-cycle patterns that include $p3$: A=1 B=1 in the second cycle can detect the defect.

The UCA proposed in this chapter performs this structural analysis on every cell, by injecting every potential defect and for every cell pattern checking which paths inside the cell are active. Every (d, p) tuple is analyzed to determine if it is logically undetectable or logically detectable. A logically detectable defect is:
(1) A short that connects V_{DD} to V_{SS} and propagates the fault through the following compartments to the cell's output.
(2) An open that is part of an active path in the cell.
If these conditions are not met the defect-pattern tuple is logically undetectable and its simulation can be avoided.

5.3 Undetectability Checker Algorithm

As seen in the previous sections, a cell-internal defect d cannot be detected by a cell pattern p if it does not impact on the cell's logic behavior or dynamic properties. This chapter presents a *Undetectability Checker Algorithm* (UCA) that automates the structural analysis shown in section 5.2 to evaluate if a (d, p) tuple is logically undetectable. Every CMOS cell can be seen as a group of nets, interconnected by parasitics and transistors. Because of that, representing a cell with a graph is very convenient. In fact, every net can be seen as a node and every parasitic component or transistor can be seen as an edge connecting two nodes (nets) together. The proposed UCA uses the *NetworkX Python* package to generate a graph for each cell and to analyze if an injected defect d is



(a) $p0$: A=0 B=0, logically undetectable.

(b) $p1$: A=0 B=1, logically undetectable.

(c) $p2$: A=1 B=0, logically detectable.

(d) $p3$: A=1 B=1, logically undetectable.

Figure 5.5: AND cell with short defect (in red) and all the cell patterns.

logically detectable by a cell pattern p or not.

As shown in the pseudo-code in Algorithm 5.1, UCA analyzes every cell c in the library C. To do so it starts from the cell's *Detailed Standard Parasitic Format* (DSPF) [Cadence Design Systems, 2018] file. From this file all the information about the cell's nets, transistors and parasitic components is extracted. For every cell pattern $p \in P_c$ a graph is generated. The active transistors are inserted in the graph as edges between two nodes, where each node is a net. Thus, a compartment output in c is always connected to V_{DD} or to V_{SS}. To identify the active path from cell inputs to outputs, we analyze cell compartments sequentially from cell inputs until all cell outputs have a connection to V_{DD} or to V_{SS}. At the end of this process a graph $G[c;p]$ is generated for a given cell c and for every cell pattern $p \in P_c$. When the defect-free representation of the cell is

available all the potential defects are injected one by one to evaluate their effect on the cell's behavior.

As described in Line 06 and Line 12 of Algorithm 5.1, each short defect $d \in D_c$ is injected in the graph as an edge connecting two nodes (nets) together and each open is injected by removing an edge between two nodes. After this has been done the effect of the defect is evaluated by checking if there is any path connecting V_{DD} to V_{SS} in case the defect is a short, or if the output is floating in case the defect is an open. Only an open defect that is included in an active path in the cell can disconnect the cell's output from V_{DD}/ V_{SS}. If the condition in Line 07 or in Line 13 is met, the (d, p) tuple is logically detectable, otherwise it is logically undetectable. The output of the UCA is a set of logically undetectable (d, p) tuples that can be excluded from simulation.

Algorithm 5.1: DETECTABILITY CHECKER ALGORITHM

```
01: for all c ∈ C do {
02:   for all p ∈ P_c do {
03:     G[c;p] = Graph_{c,p};          // Identify the connected nets per compartment
04:     for all d ∈ D_c do {
05:       if d is a short defect then {
06:         F_d = G[c;p] ∪ {d};
07:         if ∃ path from V_DD to V_SS in F_d then {
08:           (d,p) is logically detectable; }
09:         else { (d,p) is logically undetectable; }
10:       }
11:       else {                       // d is an open defect
12:         F_d = G[c;p] \ {d};
13:         if ∄ path from V_DD/V_SS to an output of c then {
14:           (d,p) is logically detectable; }
15:         else { (d,p) is logically undetectable; }
16: } } } }
```

5.4 Experimental Results on GPDK045 Library

The proposed UCA was applied on 476 cells from the GPDK045 library with 30,138 potential short defects and 62,771 potential open defects, a total of 5,385,168 (d, p) tuples were analyzed. The characterization optimized by UCA is compared to the regular characterization of short defects using one-cycle patterns and open defects using two-cycle patterns. Our total defect characterization time is the sum of UCA run time and

the characterization time only for the logically detectable (d, p) tuples.

Performed by Cadence's tool Modus v19.1, the regular characterization of GPDK045's 476 cells for ∼30k short defects using one-cycle patterns took 493 hours. Performing the structural analysis with UCA on all the cells and the potential shorts took only 3 hours and 15 minutes and it identified 55% logically undetectable (d, p) tuples. By excluding their simulation the characterization time significantly drops from 493 hours to only 217 hours. So running the characterization preceded by UCA would take a total of 220 hours against the 493 hours of the regular CAT characterization, which is a 55% time reduction.

The results obtained comparing the new method against the characterization of the 476 cells for ∼60k open defects with two-cycle patterns are similar. The regular characterization took 1,462 hours. Running the UCA on every cell and their potential open defects took 2 hours and 53 minutes. 35% of the (d, p) tuples were identified as logically undetectable and the estimated remaining characterization time drops to 816 hours. In total the characterization preceded by UCA would take 819 hours against the 1,462 hours of the regular characterization. Figure 5.6 shows the graph with the regular characterization time vs the UCA characterization time, both for shorts and for opens. Considering the overall result, the characterization time can be reduced from a total of 1,955 hours ($\sim$ 81 days) to 1,039 hours ($\sim$ 43 days), that is a significant 47% reduction in characterization time.

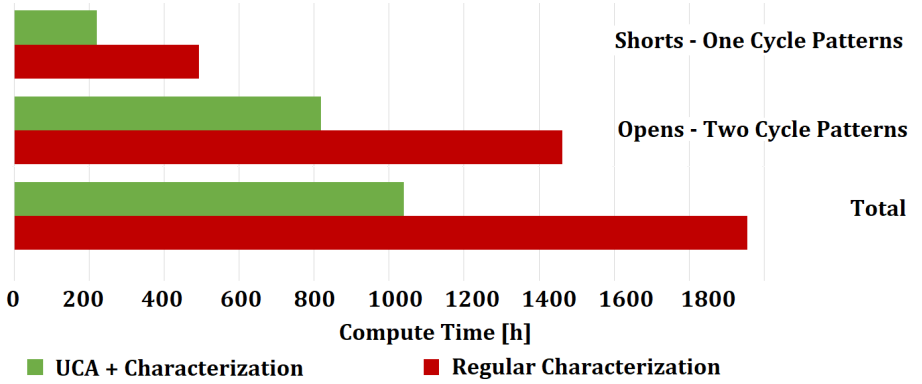
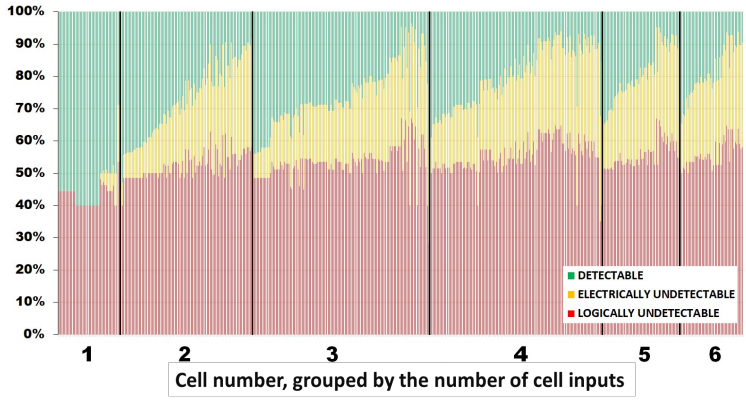


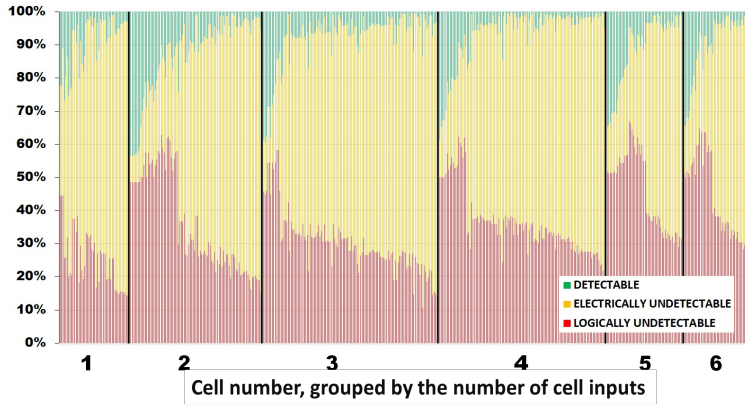
Figure 5.6: Regular characterization vs. characterization preceded by DCA.

Furthermore UCA results can be compared to the simulation results, to gather additional information. There are four possible outcomes:

(1) UCA predicts (d, p) to be logically undetectable and the simulation confirms the undetectability. All the UCA's logically undetectable tuples were confirmed by the sim-



(a) Shorts - one cycle patterns.



(b) Opens - two cycle patterns.

Figure 5.7: Detectability of (d, p) tuples for the GPDK045's libary cells.

ulations.

(2) UCA predicts (d, p) to be logically undetectable, but the simulation can detect it. In this case there is an estimation error by UCA. No errors were noted for the analyzed cells and UCA was always correct.

(3) UCA predicts (d, p) to be logically detectable, but d does not cause a fault effect at any cell output, i.e. the simulation cannot detect it. In this case the tuple is electrically undetectable, shown as Figure 5.3.

(4) UCA predicts (d, p) to be logically detectable and the simulation can detect it. In this case the tuple is detectable, shown as Figure 5.3.

Figure 5.7 shows the results collected with this investigation. The cells are sorted based on the number of cell's inputs, from low to high. Every group is internally sorted based on the drive strength. Given a number of cell inputs, more parallel transistors are used to implement a higher drive strength. These results show that the percentage of electrically undetectable (d, p) tuples grows with the cell's drive strength. This is because these cells have many parallel branches that can guarantee the correct functioning despite one branch affected by the defect. In fact even if one of the active paths in the cell is interrupted by an open or opposed by a short, there are many alternative paths that can still guarantee the correct cell functioning.

Figure 5.7 shows the results collected for hard defects ($1m\Omega$ shorts and $1G\Omega$ opens). The detectable (d, p) tuples can become electrically undetectable when the defect becomes weaker. Equation 5.1 shows that when a short-defect resistance is larger than a certain value, it does not cause a fault. For an open defect, if its resistance is smaller than a certain value, the open defect cannot cause a serious delay that can be detected. What we would see is a decreasing percentage of detectable tuples in favor of an increasing percentage of electrically undetectable tuples, as the defect becomes weaker. The results are shown in Section 6.3.

5.5 Conclusion

CAT library characterization is a very time consuming process, since for every cell c, it requires to simulate each (d, p) tuple. Most of the tuples are undetectable and an efficient structural analysis can accurately determine if pattern p can detect defect d independently from the electrical properties of the cell. This can be exploited to omit the corresponding simulation, reducing the characterization time. This chapter proposes a *Undetectability Checker Algorithm* (UCA) that automates this analysis. The results obtained running UCA on the GPDK045 library show that running it before characterization can significantly reduce the number of simulations and save approximately 47% of characterization time. The UCA is also a valuable tool to analyze regular CAT characterization results, identifying electrically undetectable (d, p) tuples.

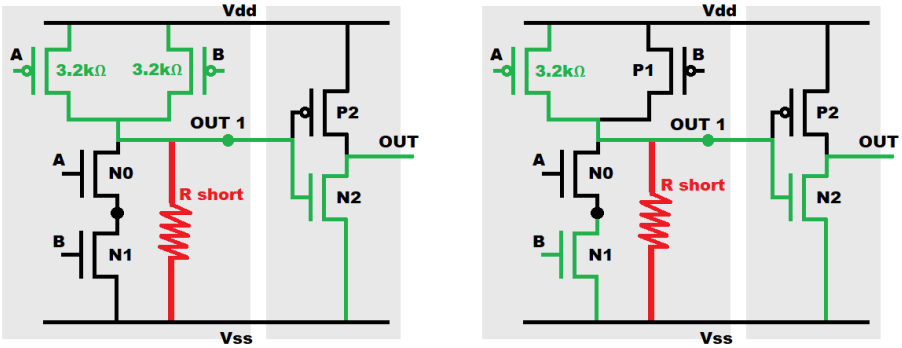
CHAPTER 6

Optimization of Cell-Aware Weak-Fault Coverage

It is widely accepted in the IC test community that manufacturing defects are accurately modeled as resistive shorts and opens [Bushnell and Agrawal, 2000]. In case of a low-ohmic short or a high-ohmic open, we classify the defect as *hard*. Variants of the same defect with higher (for shorts) or lower (for opens) resistance exhibit a less impactful fault behavior than hard defects and hence are called *weak* defects. In general, the weaker the defect, the more difficult it becomes to detect. In a digital CAT, this typically implies that the number of cell-level patterns that detect an increasingly weaker defect variant diminishes until the defect is so weak that no patterns are left and the defect becomes undetectable. Per cell pattern, some defect resistance exists as a tipping point for a defect to change from detectable to non-detectable. We referred this defect resistance as the *critical resistance*.

The defect detectability analysis in Section 5.2 helps to explain why different cell patterns can detect the same defect up to different critical resistances. Short-defect detectability relies on the voltage division along the shorted path from power to ground (i.e. from V_{DD} to V_{SS}). The voltage division is determined by the short-defect and transistor resistances involved in the shorted path. Different cell patterns switch on and off different groups of transistors, and hence affect the transistor resistances involved in the

shorted path. Given a certain short-defect resistance, different patterns cause different voltage divisions, and hence result in different detection results. Therefore, different patterns can have different critical resistances. Open-defect detectability is determined by the delay measured at cell outputs. The cell delay is affected by the *RC* network that is activated by a cell pattern applied at cell inputs, see the analysis in Section 3.2. Different cell patterns cause different $R_{\text{open}}C$ values which lead to different delays, and therefore result in different detection results. As C varies with applied cell patterns, given a threshold delay, the critical R_{open} changes per pattern. For example, using Equation 5.1 and assuming $V_{TH} = \frac{V_{DD}-V_{SS}}{2}$ where V_{TH} is the threshold voltage for logic 1, the short defect in Figure 6.1 is detectable by cell pattern A=0 B=0 until its resistance R_{short} increases to $\frac{3.2k\Omega}{2} = 1.6k\Omega$ shown in Figure 6.1(a). It is detectable by cell pattern A=0 B=1 if $R_{short} < 3.2k\Omega$ shown in Figure 6.1(b). The critical resistance is weaker when the part of the cell acting against the defect is weaker. If for example we assume the defect has a resistance equal to $2k\Omega$, p_0: A=0 B=0 will not detect it (the defect-pattern tuple is electrically undetectable), but p_1: A=0 B=1 will detect it.



(a) $p0$: A=0 B=0 (b) $p1$: A=0 B=1

Figure 6.1: AND cell with two logically detectable (d, p) tuples.

The concept of relationship between defect weakness and cell pattern count is shown in Figure 6.2. Assuming the defect in Figure 6.2 is a short defect, along the axis from left to right, the defect resistance, and also weakness increases. Given a defect, for all cell patterns, the defect variant corresponding to the weakest critical resistance is referred as the *weakest fault*. In Figure 6.2, three cell patterns can detect the short defect. Each of them covers different resistance range, which is referred as *detection interval*. Pattern *p1* can detect the short defect varying from 0Ω up to 40kΩ which is the weakest *critical resistance* compared with the other cell patterns. Therefore, 40kΩ short variant is the weakest fault, and [0, 40kΩ] is the detection interval of this short defect.

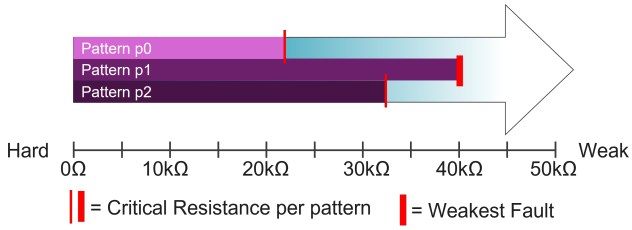
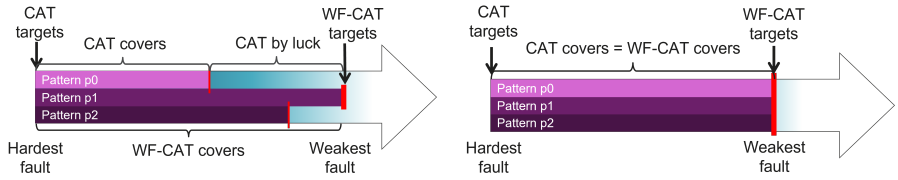


Figure 6.2: The number of cell patterns detecting a defect reduces as it becomes weaker.

In this chapter, we propose a cell-aware test generation approach that explicitly targets the weakest fault at each potential cell-internal defect location, referred to as weakest-fault CAT (WF-CAT). Detecting the weakest fault means all defect variants in the corresponding detection interval are also covered. To analyze the benefit from WF-CAT, compared to general CAT, we classify defects into two categories: (1) *resistance-dependent faults* and (2) *resistance-independent faults*, shown in Figure 6.3. Given a defect, if its cell patterns have different detection intervals, we refer to this defect as a resistance-dependent fault. General CAT normally targets the hardest defect variant. For the fault in Figure 6.3(a), the cell-aware ATPG engine selects one of the three patterns to expand to chip level, and covers the detection interval of that expanded cell pattern. As general CAT has no insight of the detection-interval difference among the three cell patterns, it only guarantees covering the smallest detection interval. Instead, WF-CAT targets the weakest defect variant to cover the largest detection interval. Therefore, WF-CAT detects more weak defect variants. However, for resistance-independent faults, CAT and WF-CAT achieve the same test quality, shown in Figure 6.3(b). If all cell patterns' detection intervals are identical, we refer to this defect as resistance-independent faults.



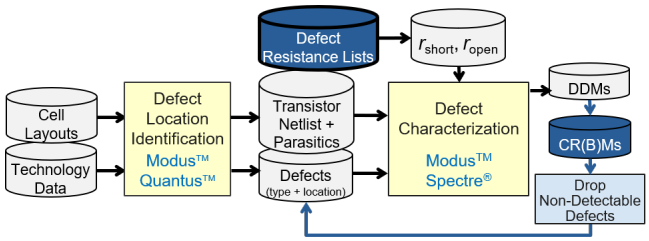
(a) Resistance-dependent fault.

(b) Resistance-independent fault.

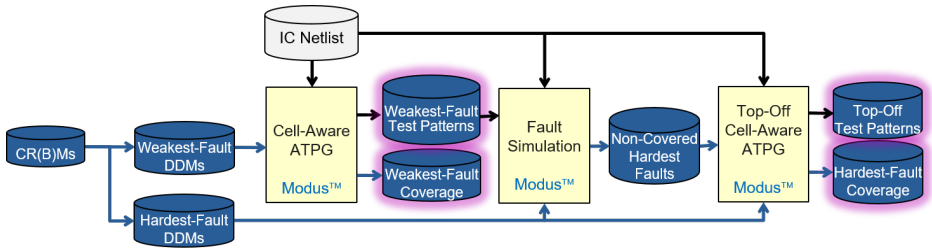
Figure 6.3: Comparison of general CAT and WF-CAT on two types of faults.

We implemented this WF-CAT approach in an experimental scripted tool flow, using various functions of Cadence's commercially-available Modus ATPG tool (release 19.1) as building blocks (see Figure 6.4). The dark-blue items in this figure are the new ele-

ments in this tool flow. In Step 1, per library cell, we identify the weakest still-detectable resistances for all defect-pattern combinations; we store them in a *critical resistance matrix* (CRM). In Step 2, these CRMs are used by the cell-aware ATPG engine to explicitly target the weakest still-detectable faults and generate test patterns for them.



(a) Step 1: Defect characterization (once per library).



(b) Step 2: Weakest-fault cell-aware ATPG (once per design).

Figure 6.4: Weakest-fault cell-aware test (WF-CAT) tool flow (legend: □ = CAT functions; □ = CAT files; ■ = WS-CAT files; ■ = WS-CAT results).

The remainder of this chapter is organized as follows. Section 6.1 provides the related prior work. Section 6.2 introduces the library characterization flow of our proposed approach, in which now we simulate two ranges instead of two single resistance values. Characterization results for a 45nm CMOS standard-cell library are presented in Section 6.3. Section 6.4 describes our WF-CAT ATPG flow of which the explicit objective is to improve the weakest fault coverage, while maintaining the original (hard) fault coverage of the conventional CAT approach. Section 6.5 presents weakest-fault ATPG results for eleven benchmark circuits and Section 6.6 concludes this chapter.

6.1 Related Prior Work

Virtually all prior work on testing for resistive open and/or short defects does not relate to CAT [Shinogi et al., 2001, Mohammadat et al., 2015, Arai et al., 2017, Andjelkovic et al., 2018, Medeiros et al., 2019, Karel et al., 2019], while prior work on CAT almost exclusively targets hard defects [Hapke et al., 2009, Hapke et al., 2014b, Singh, 2016, Howell et al., 2018, Dixit et al., 2018]. To the best of our knowledge, the only exception is a paper by Hapke et al. [Hapke et al., 2010], in which CAT is used to try to detect both hard and weak short defects. The authors treat variants of the same defect with different resistive values as independent defects if their corresponding cell-level test patterns sets are non-identical. For a short defect s, consider two variants s_1 and s_2 with s_2 weaker than s_1. In case $ps_c(s_1) = ps_c(s_2)$, [Hapke et al., 2010] treats these two defect variants as one defect, but if $ps_c(s_2) \subset ps_c(s_1)$, this single defect is modeled in [Hapke et al., 2010] as two independent defects. If short defects are characterized for n different resistance values, a single defect can end up in the fault model of [Hapke et al., 2010] as anywhere from one up to n independent faults. We consider these unpredictable effects on fault counts, and consequently on fault coverages, as undesirable. We ensure that a defect is only considered once.

Preliminary experiments on 324 cells from Cadence's GPDK045 45nm CMOS library [Cadence Design Systems, 2014] show that if we vary the resistance value of the short defects in these library cells from 0 to 50kΩ, for about 60% of the short defects the cell pattern set capable of detecting the defect is independent of the short's resistance value. However, for the remaining 40% of the shorts, the cell pattern set size varies as a function of the resistance value. If we consider per defect, the hardest and the weakest still-detectable resistance values, the defect model proposed in [Hapke et al., 2010] would inflate the total number of short defects $(11,394)$ by a factor $0.6{+}2{\times}0.4 = 1.4$ to $15,907$ 'defects'. Considering 13 resistance values in the range $[0, 30k)\Omega$ (see Section 6.3), this unjustified inflation would even grow to $1.7 \times 11,394 = 19,363$ 'defects'.

Hapke et al. [Hapke et al., 2010] only discuss shorts, but in this chapter we address both short and open defects. We consider a weaker defect at the same location as just another variant of the same defect, thereby keeping the total defect count of a library cell a property of that cell only, not dependent on the number of resistance values considered during defect characterization. To avoid the anomalous fault coverage definition from [Hapke et al., 2010], we introduce a new weakest-fault coverage *wfc*, next to the regular ('hard') fault coverage (*fc*).

6.2 Library Characterization

This section describes the WF-CAT library characterization, shown as Step 1 in Figure 6.4. We assume that a defect d with resistance $r = R$ which is detectable by cell pattern p, remains detectable by p for all harder variants of d (i.e., for shorts with $r \leqslant R$ and for opens with $r \geqslant R$). This assumption implies that for each defect-pattern (d, p) combination for which there is a resistance value for which d is detectable by p, there is a weakest still-detectable resistance, which we refer to as the *critical resistance* of that particular defect-pattern combination.

Step 1 of our tool flow determines CRMs for a standard-cell library C, for further usage in Step 2. On the basis of the cell layout and transistor-level netlist with extracted parasitics, we first identify for each library cell $c \in C$ its set of potential defects D_c (type and location), as described in Chapter 3. Subsequently, for each defect-pattern tuple (d, p) with $d \in D_c$ and $p \in P_c$, where P_c denotes the exhaustive cell-level pattern set for cell c, we determine its critical resistance $CRM(d, p)$. This can be implemented through an iterative loop of Modus' defect characterization function *DC* over appropriate ranges of resistance values. For a user-specified pair of short/open defect resistances r_{short}/r_{open}, *DC* determines for all defects $d \in D_c$ and all cell patterns $p \in P_c$ whether or not p detects d, denoted by a binary variable $DDM(d, p)$. We could simply repeat DC for all possible resistance values in the defect resistance range considered. As Figure 6.5 shows, the resulting binary DDMs can be condensed into a single CRM, where each entry corresponds to the weakest resistance value r_{short} or r_{open} for which any of the DDMs denoted detection at (d, p).

DC is a compute-intensive operation. Fortunately, note that DC needs to be performed only once for each library release, while its results can be reused for all chip designs based on that library. This implies that we can afford to spend quite some compute time

Patterns	Short=0.1Ω		Open=1GΩ	
	d1	d2	d3	d4
p0		1		
p1	1	1		1
p2	1		1	
p3		1	1	1

(a) Multiple DDMs.

Patterns	Short		Open	
	d1	d2	d3	d4
p0		1k		
p1	5k	5k		70M
p2	1k		10M	
p3		10k	10M	1G

(b) Critical resistance matrix (Ω).

Figure 6.5: Multiple DDMs for various open/short defect resistance value pairs can be condensed into a single CRM.

on DC. However, even a single DC iteration is expensive with respect to compute time, due to the fact that it repeats a detailed analog simulation in three nested loops: for all cells, for all defects, and for all cell patterns. Adding a fourth loop 'for all resistance values' forces us to consider only a limited number n of defect resistances to avoid the required compute time to become excessively large.

First we determine the defect resistance ranges to be considered. The hardest defect resistance of each range should be *super hard*, such that we maximize the pattern set for each defect. The weakest resistances of each range should be such that for each defect-pattern combination, its critical resistance is included in the range. Hard shorts are low-ohmic and thus the natural candidate for the super-hard short is 0Ω. This is what we did in previous chapters. For super-hard opens and both weakest shorts and opens, the resistance values can vary per technology node and standard-cell design.

Next, the user needs to determine the number n of defect resistance pairs to be considered, and what their values will be. In our experience, a value of n for which the cumulative DC compute time is still considered affordable is too small to accurately perform DC for all possible critical resistance values; even at a granularity of, say, 1kΩ. Thus we split both defect resistance ranges into n *bins* that will approximate the critical resistance values.

Modus' DC function needs as inputs (1) the transistor-level netlists of a set library cells C, (2) the set of short and open defects D_c for all $c \in C$, and (3) the resistance values r_{short} and (4) r_{open} for short and open defects, respectively. Its output is a binary DDM_c for all $c \in C$.

In WF-CAT, defect characterization consists of n iterations of Modus' DC function. It approximates for each library cell c the critical resistances for each defect-pattern combination (d, p) out of the iteratively generated $DDM_c[i]$ (for $i \in [1..n]$) in a *critical resistance bin matrix CRBM*. If we order the resistance value pairs for the subsequent DC iterations from hard to weak (i.e., resistances increasing for shorts and decreasing for opens), then $CRBM_c$ is defined as follows:

$$CRBM_c(d,p) = MAX_{i \in [1..n]}\{i | DDM_c[i](d,p) = 1\}$$

Note that CRBM store DC iteration numbers, instead of actual resistances values; this allows us to treat short and open defects in an identical way. Translation from iteration numbers to user-defined defect resistance values is done through a two-dimensional look-up table $R[t, i]$, where defect type $t \in \{short, open\}$ and DC iteration $i \in [1..n]$.

The n resistance values that determine the critical resistance bin boundaries do not need

to be equidistant. It is best if these n values are chosen such that the division of the defects over the critical resistance bins is balanced.

Algorithm 6.1 presents our MULTI-R defect characterization algorithm, which iteratively calls Modus' DC function to determine CRBMs for all cells in a library. Exploiting the fact that for subsequent DC iterations, the defect resistance values are ordered from hard to weak, we can for a defect-pattern combination (d, p) which in iteration i is still detectable simply overwrite the critical resistance bin so far by assigning $\mathit{CRBM}(d, p) = i$ (Line 09). We can save significant compute time by pruning the number of defect simulations needed. Exploiting our assumption that all defects harder than the critical resistance remain detectable by the same cell pattern, as soon as for a particular iteration i a defect $d \in D_c$ is no longer detectable for any pattern $p \in P_c$, we can drop that defect from subsequent DC iterations (Line 10). The simulation can be sped up further by dropping cell patterns if a cell pattern no longer detects d. However, the current version of the tool does not allow it. The generation of the full set of cell patterns for d is packed inside the tool, and we cannot change it. Similarly, if in a DC iteration a cell $c \in C$ no longer has any detectable defects left, we can exclude that cell from subsequent DC iterations (Line 13).

From the CRBMs that are generated by the WF-CAT library characterization, we extract

Pattern	**Short**		**Open**	
AB/Y	d1	d2	d3	d4
p0 = 00/L		4		
p1 = 01/L	6	6		2
p2 = 10/L	4		4	
p3 = 11/H		10	4	1

(a) Critical resistance bin matrix.

Pattern	**Short**		**Open**	
AB/Y	d1	d2	d3	d4
p0 = 00/L				
p1 = 01/L	1			1
p2 = 10/L			1	
p3 = 11/H		1	1	

(b) Weakest-fault DDM.

Pattern	**Short**		**Open**	
AB/Y	d1	d2	d3	d4
p0 = 00/L		1		
p1 = 01/L	1	1		1
p2 = 10/L	1		1	
p3 = 11/H		1	1	1

(c) Hardest-fault DDM.

Figure 6.6: Example CRBM and corresponding WF-DDM and HF-DDM.

Algorithm 6.1: MULTI-R DEFECT CHARACTERIZATION

Inputs : C, D_c for all $c \in C$, n, $R[\{short, open\}, \{1..n\}]$;
Outputs : $CRBM_c$ for all $c \in C$;
01: $i = 1$;
02: **while** $i \leq n \wedge C \neq \varnothing$ **do {**
03: $DDM_c[i]$ = *DC*$(C, D_c, R[short, i], R[open, i])$;
04: **for all** cells $c \in$ library C **do** {
05: generate exhaustive cell-pattern set P_c with $|P_c| = \#out_c \times 2^{\#in_c}$;
06: **for all** defects $d \in$ defect set D_c **do** {
07: *detectable* = FALSE;
08: **for all** patterns $p \in$ pattern set P_c **do** {
09: **if** $DDM_c(d, p) = 1$ **then** *{*
10: *detectable* = TRUE;
11: $CRBM_c(d, p) = i$;
12: } };
13: **if** *detectable* = FALSE **then {** $D_c = D_c \setminus \{d\}$ **}**;
14: **if** $D_c = \emptyset$ **then {** $C = C \setminus \{c\}$ **}**;
15: }
16: $i = i + 1$;
17: }

where Modus' function DC has the following role:

Inputs : C, D_c for all $C \in C$, r_{short}, r_{open};
Outputs : DDM_c for all $c \in C$;
21: **for all** cells $c \in$ library C **do** {
22: generate exhaustive cell-pattern set P_c with $|P_c| = \#out_c \times 2^{\#in_c}$;
23: **for all** defects $d \in$ defect set D_c **do** {
24: **for all** patterns $p \in$ pattern set P_c **do** {
25: **if** *FaultSimulation*$(c + d, p) \neq$ *FaultSimulation*(c, p)
26: **then** $DDM_c(d, p) = 1$;
27: **else** $DDM_c(d, p) = 0$;
28: } } }

two DDMs that play a prominent role in our WF-CAT ATPG approach (see Section 6.4). They are the weakest-fault DDM (WF-DDM) and the hardest-fault DDM (HF-DDM). This is illustrated by means of a small example in Figure 6.6. Figure 6.6(a) shows a CRBM corresponding to the CRM in Figure 6.5(b), and the tables in Figures 6.6(b) and 6.6(c) show respectively the extracted WF-DDM and HF-DDM. The HF-DDM denotes detection ('1') for all defect-pattern combinations for which the CRBM has a critical resistance recorded. In fact, the HF-DDM corresponds to the DDM as generated in DC iteration 1 of our MULTI-R DC algorithm and is based on two single defect resistance values for all detects: one for shorts and one for opens. This is different for the WF-DDM. The WF-DDM denotes detects only for those cell patterns that detect the weakest still-detectable variant of a defect. As the weakest still-detectable variant is defined per defect, the actual resistance values can vary for each defect. Note that all faults have at least one cell pattern that detects the weakest fault, but there can also be multiple cell patterns that achieve this, as is the case for defect $d3$ in the example in Figure 6.6(b).

6.3 Library Characterization Results

We performed library characterization for 324 combinational standard cells from Cadence's GPDK045 45nm CMOS library [Cadence Design Systems, 2014]. We used Modus v19.1, assisted by Quantus v18.1 for parasitic extraction during defect location identification and assisted by Spectre for analog simulations during defect characterization. We extent the experiments of one-cycle cell-level pattern detection in [Hu et al., 2020] by including simulation of two-cycle patterns. We first illustrate our approach with one-cycle pattern simulation by Spectre v19.1, afterward, report the results related to two-cycle detection by Spectre v20.1.

6.3.1 Simulation of One-Cycle Cell Patterns

For short defects, we set the defect resistance range to [0Ω, 50kΩ); 0Ω is a rather obvious choice for a super-hard short, and through simulation we found that no short defect $\geq$50kΩ was detectable in any of the library cells. For open defects, initially, we considered to follow [Hapke et al., 2010] and use 1GΩ as super-hard resistance. However, we found some defects that required an open resistance larger than 1GΩ to become detectable; hence we used 1,000GΩ = 1TΩ as safe super-hard resistance for opens. Finally, we ended up using 13 resistance bins; their resistance values are shown in Figure 6.7.

We identified 50,541 defect locations for the 324 library cells, i.e., on average 156.0 defects per cell. We performed defect characterization on these library cells with the

abovementioned 13 defect resistance pairs using Algorithm 6.1. Figure 6.7 shows the number of defects considered per DC iteration. In the first iteration, all 50,541 defects were simulated for super-hard resistance values $r_{\text{short}} = 0\Omega$ and $r_{\text{open}} = 1T\Omega$. This required 18.1 days CPU time on high-end compute servers at Cadence in Endicott, NY, USA. 57.6% of the defects were classified as *non-detectable* by one-cycle cell patterns. The remaining 42.4% of the defects that were detectable in Iteration 1 continued in the second DC iteration with $r_{\text{short}} = 1\Omega$ and $r_{\text{open}} = 75M\Omega$. Figure 6.7 shows for all DC iterations the number of defects considered. With increasingly weaker defects in the subsequent DC iterations more defects become undetectable and once that happens, they are dropped from further DC iterations (as described in Algorithm 6.1). The overall compute time for MULTI-R DC was 62.6 days; quite expensive, but thanks to the defect and cell dropping mechanisms in Algorithm 6.1, the defect characterizations of 12 additional resistance values for all defects increased the compute time only 2.5× with respect to the first iteration; this is a 73.4% reduction from what one would expect for $13 \times 18.1 = 235.3$ days. In our case, we were fortunate that the MULTI-R defect characterization could be performed on a multi-processor cluster, such that the elapsed wall-clock time was only 21 days.

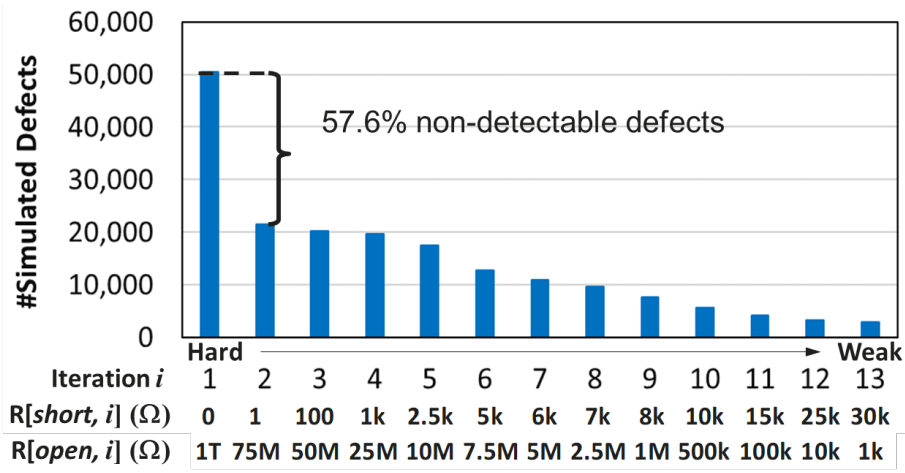


Figure 6.7: The number of defects per defect characterization iteration.

We defined pattern set $ps_c(d)$ as the set of cell patterns that detect defect d in cell c. On the basis of this definition, we can now formulate the definitions for two specific pattern sets: $hps_c(d)$ contains all cell patterns that are able to detect the hardest variant of defect d, while $wps_c(d)$ contains all cell patterns that are able to detect the weakest still-detectable variant of defect d.

$$hps_c(d) = \{p \in P_c | DDM_c[1](d,p) = 1\}$$
$$wps_c(d) = \{p \in P_c | CRBM_c(d,p) = MAX_{p \in P_c}(CRBM_c(d,p))\}$$

Figure 6.8 shows both $|wps_c(d)|$ and $|hps_c(d)|$ for all 21,438 detectable defects in our library of 324 cells. By definition, $wps_c(d) \subseteq hps_c(d)$, and thus $|wps_c(d)| \leq |hps_c(d)|$. 66.9% of all detectable defects are *resistance-independent faults*, as their detecting pattern set does not depend on the defect resistance. For these defects $|wps_c(d)| = |hps_c(d)|$ and hence $wps_c(d) = hps_c(d)$. For these defects, regular CAT and WF-CAT achieve the same test quality. The other 33.1% of all detectable defects are *resistance-dependent faults*, for which $wps_c(d) \subset hps_c(d)$ and hence $|wps_c(d)| < |hps_c(d)|$.

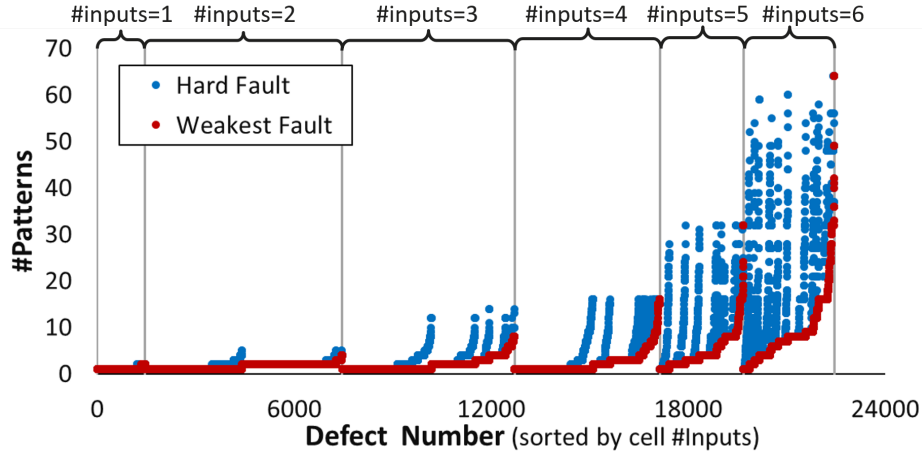


Figure 6.8: The number of cell patterns for the hard and weakest faults.

To detect the weakest variants of those defects, it is important to select the right cell patterns. For all resistance-independent faults in the 324 GPDK045 library cells, on average $|hps_c(d)| = 8.9$ patterns, while $|wps_c(d)| = 3.5$ patterns. An ATPG tool that is unaware of these differences has a (8.9-3.5)/8.9=60.7% probability to select a cell pattern for expansion that does *not* detect the weakest fault(s). The maximum pattern set size difference $|hps_c(d)| - |wps_c(d)|$ is 56 patterns for a defect in cell AOI33X1.

For each cell, we determined the required minimal number of cell patterns that can cover all super-hard faults respectively all weakest faults. This problem is equivalent to the well-known $\mathcal{NP}$-hard set cover problem and addressed by heuristic algorithm MINCOVER in Chapter 4. In general, hard faults can be detected by more cell patterns than weakest faults. Consequently, as shown in Figure 6.9, for each library cell more

cell patterns are required to cover all weakest faults than to cover all hard faults. Over all characterized 324 cells, the accumulated required minimal number of cell patterns to cover all hard faults is 1,665, while 384 additional patterns (= +23%) are needed to cover all weakest faults. On average, we require about one more cell pattern per library cell to cover all the weakest faults than we do to cover all hard faults.

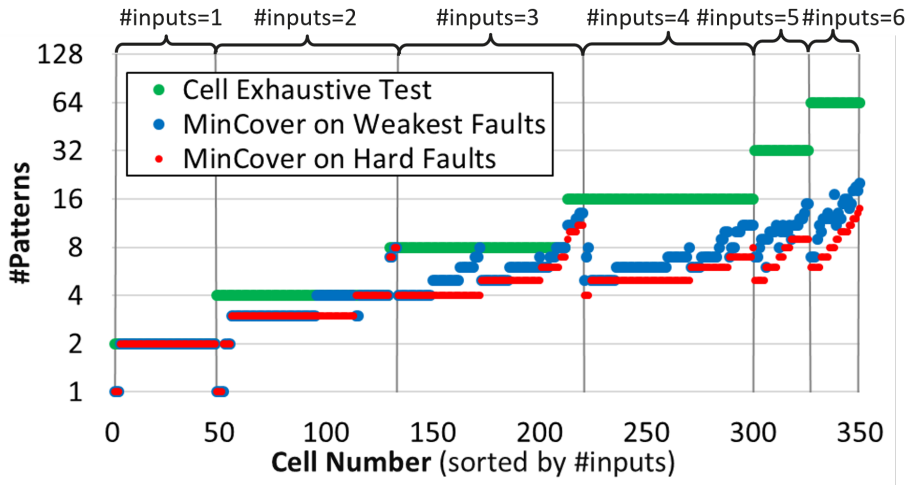


Figure 6.9: The minimal required number of cell patterns to cover all hard or weakest faults, compared to the cell-exhaustive pattern set.

6.3.2 Simulation of Two-Cycle Cell Patterns

To run the defect characterization, we use Modus v21.1 invoking Spectre v20.1 which is much faster than the older versions.

For total 28,688 open defects in the 324 cells, we perform simulation with two-cycle cell patterns. For a cell with n inputs, the total number of two-cycle patterns is $2^n \times (2^n - 1)$ excluding the pattern pairs with identical patterns in their two cycles. Expanding the cell-level two-cycle cell patterns with transitions at multiple input bits requires these signal transitions arriving at the cell inputs simultaneously. This requirement is difficult to satisfy in a circuit design. Also the number of exhaustive two-cycle cell patterns increases drastically with the number of cell inputs. To generate cell patterns with high probability of being expanded by ATPG and to reduce the simulation time, we reduce the set of exhaustive two-cycle cell patterns to a smaller set which includes the two-cycle patterns with only one input transition and causing at least one output transition.

We set 10ns as the largest delay tolerance. If for a two-cycle pattern p, an open defect d causes a delay, compared with nominal cell delay, larger than 10ns, we record $DDM(d, p) = 1$. The used ten open resistances are shown in Table 6.1. By removing the non-detectable defects per iteration, we reduce the simulation CPU time from 28 to 11 days, which constitutes 61% decrease. The simulation results show that 4,176 (= 29.5%) out of 14,148 detectable open defects are resistance-dependent faults.

	Hard									**Weak**
Iteration i	1	2	3	4	5	6	7	8	9	10
$R[\text{open}, i]$ (Ω)	1G	75M	50M	25M	10M	7.5M	5M	2.5M	1M	1k

Table 6.1: Open-defect resistances used to split the resistance range.

6.4 Weakest-Fault Cell-Aware ATPG

Cell-aware ATPG uses as inputs a cell-level netlist of the circuit-under-test and DDMs of all library cells instantiated in the netlist. For each cell instance, all detectable defects in the corresponding DDM are faults and serve as targets of the ATPG process. The ATPG tool tries to expand cell patterns from cell to chip level to cover as many as possible cell-internal faults. The circuit design surrounding a target cell might prevent the ATPG tool to successfully expand a cell pattern. The gate-exhaustive ATPG in Section 2.7 shows that the chance to successfully expand an arbitrary cell pattern is $\sim$71%; and this is confirmed by what is reported as 'gate-exhaustive fault coverage' in [Hapke et al., 2011b].

At the end of Step 1, library characterization, we have generated two sets of DDMs for all library cells: WF-DDMs and HF-DDMs. The two DDMs have the same matrix dimensions, i.e., their defect/fault columns and cell pattern rows are identical. WF-DDMs and HF-DDMs only differ with respect to resistance-dependent fault columns, as the cell pattern sets of these faults in the WF-DDM matrix are true subsets of their HF-DDM counterparts. Modus' cell-aware ATPG can be executed on the basis of either WF-DDMs or HF-DDMs.

ATPG with WF-DDMs exclusively detects cell-internal faults only by means of those cell patterns that are able to detect the weakest still-detectable defect variant. With only WF-DDMs, we simply do not inform Modus that other cell patterns might be able to detect harder variants of the same defect. Cell-aware ATPG on the basis of WF-DDMs reports the fault coverage as the *weakest-fault coverage* (*wfc*). The *wfc* will end up lower than the regular fault coverage *fc* based on HF-DDMs, as the WF-DDMs contain fewer

alternative cell patterns that each offer a chance to detect the target fault by a successful expansion to chip level. For example, in Figure 6.6, the column *d2* in the WF-DDM has only one cell pattern, while in the HF-DDM has three cell patterns. To detect *d2*, the ATPG based on WF-DDM can only expand pattern *p3*. Once *p3* is expanded successfully, not only the weakest defect variant is detected, but also the defect variants between the hardest and the weakest resistances are all detected. However, *p3* probably cannot be expanded successfully. The ATPG with HF-DDM has three alternative cell patterns to expand to detect *d2*. This increases the probability of detecting *d2*, but cannot guarantee the expansion of *p3*, and hence cannot guarantee detecting the maximum defect interval. The benefit of WF-CAT ATPG is that by explicitly targeting weakest-faults, *wfc* is expected to be significantly higher than when cell-aware ATPG is performed on the basis of HF-DDMs only.

As shown in Step 2 in Figure 6.4, our WF-CAT ATPG tool flow consists of three stages. First, we focus on detection of the weakest faults, as detection of a weak fault implicitly guarantees detection of all harder variants of the same defect, while the opposite is not necessarily true. This is done by executing Modus cell-aware ATPG on the basis of WF-DDMs. A first set of test patterns TP_1 is generated, along with their *wfc*. Test pattern set TP_1 certainly also already provides significant regular fault coverage *fc*, but as Stage 1 focuses exclusively on maximizing *wfc*, it is possible that *fc* is not yet maximal. Therefore, in Stage 2 we determine *fc* for TP_1 by performing fault simulation on the basis of HF-DDMs. The still undetected hardest faults are then targeted in Stage 3, in which *top-off ATPG* on the basis of HF-DDMs maximizes *fc* and generates an additional set of test patterns TP_3.

6.5 ATPG Experimental Results

We performed ATPG with Cadence's Modus v19.1 on eleven circuits [Scott Davidson, 1999, OpenCores.org, 1999, Arm, 2007] that were mapped onto Cadence's GPDK045 45nm CMOS library [Cadence Design Systems, 2014]. For each circuit, we compare the regular CAT results versus our new WF-CAT results. In our experiments, we run the regular cell-aware ATPG with the HF-DDMs based on super-hard short defects of 0Ω and super-hard open defects of $1T\Omega$. Results are presented in Table 6.3 in which circuits are sorted by increasing cell instance count. The table has six main columns: (1) circuit name, (2) design data, (3) weakest fault coverage escapes ($= 1 - wfc$), (4) hard fault coverage escapes ($= 1 - fc$), and the differences between regular CAT and WF-CAT with respect to (5) the number of chip test patterns and (6) ATPG compute time.

The column 'Design Data' consists of three sub-columns. In the first one, we list

Circuit Name	Design Data		
	$\|I\|$	F_{rd}	ΔP
b15	2,933	43.4%	4.54
b20	3,212	43.3%	3.52
aes	3,320	41.2%	4.70
b21	3,482	42.1%	3.33
dtmf	3,726	41.6%	4.70
b22	5,036	43.0%	3.42
M0	5,289	41.0%	5.74
b17	8,981	43.3%	4.58
fpu	19,136	45.7%	4.66
b18	20,545	43.9%	4.55
M3	32,626	42.2%	5.60
b19	40,298	43.6%	4.53
Avg.	12,382	43.0%	4.47

Table 6.2: Design data for twelve circuits. Circuit dtmf is from Cadence [Santosh Malagi, 2019]. Aes and fpu are from OpenCores [OpenCores.org, 1999]. M0 and M3 are from Arm [Arm, 2007]. All the other circuits are from ITC99 benchmarks [Scott Davidson, 1999].

the number of cell instances $|I|$ in the circuit. In the second sub-column, we list the resistance-dependent faults F_{rd} as fraction of the total number of faults (see Equation (6.1)). In the third sub-column we list for the faults in F_{rd} the average difference between the numbers of hardest-fault and weakest-fault cell patterns, ΔP (see Equation (6.2)).

$$F_{\mathrm{rd}} = \frac{\sum_{i \in I} |\{d \in D_{c(i)} | wps_{c(i)}(d) \subset hps_{c(i)}(d)\}|}{\sum_{i \in I} |\{d \in D_{c(i)} | hps_{c(i)}(d) \neq \emptyset\}|} \tag{6.1}$$

$$\Delta P = \frac{\sum_{i \in I} \sum_{d \in D_i} |hps_{c(i)}(d)| - |wps_{c(i)}(d)|}{\sum_{i \in I} |\{d \in D_i | wps_{c(i)}(d) \subset hps_{c(i)}(d)\}|} \tag{6.2}$$

In Equations (6.1) and (6.2), I is the set of cell instances, D_c is the set of defects for cell c, and $c(i)$ is a function that returns the cell type of cell instance i, with $i \in I$.

Columns 'Weakest FCov. Escapes (= $1 - wfc$)' and 'Hard FCov. Escapes (= $1 - fc$)' present fault coverage escape data for the weakest faults and the hard faults, each in three identical sub-columns. With the term *fault coverage escapes* we refer to the fraction of non-covered faults. The first and second sub-columns give fault coverage escapes for resp. CAT and WF- CAT, while the third sub-column lists the relative reduction of the CAT fault coverage escape percentage by the WF-CAT fault coverage escape percentage. These two columns address the main key performance indicator for ATPG: test quality. The larger the reported reduction, the more effective our WF-CAT methodology. For regular CAT, *wfc* is determined by fault simulation of the regular CAT test patterns on the

Circuit Name	Weakest FCov. Escapes (=1-*wfc*)			Hard FCov. Escapes (=1-*fc*)			Δ Chip Test Patterns	Δ ATPG Compute Time
	CAT	WF-CAT	Δ(1-*wfc*)	CAT	WF-CAT	Δ(1-*fc*)		
b15	2.00%	0.95%	-52.45%	0.49%	0.45%	-7.16%	+32.1%	+115.1%
b20	0.56%	0.23%	-58.60%	0.17%	0.17%	0.00%	+15.2%	+61.7%
aes	0.41%	0.05%	-87.91%	0.04%	0.04%	-4.29%	+13.9%	+64.8%
b21	0.62%	0.36%	-42.36%	0.29%	0.29%	0.00%	+14.7%	+60.9%
dtmf	3.35%	2.79%	-16.73%	0.99%	0.99%	-0.57%	+21.1%	+66.4%
b22	0.56%	0.27%	-52.11%	0.21%	0.21%	0.00%	+15.9%	+73.2%
M0	0.81%	0.23%	-71.66%	0.16%	0.16%	-1.07%	+25.6%	+51.6%
b17	1.79%	0.74%	-58.55%	0.44%	0.42%	-4.21%	+24.0%	+84.2%
fpu	0.29%	0.16%	-45.48%	0.14%	0.13%	-13.09%	+37.2%	+73.1%
b18	1.52%	0.61%	-59.77%	0.36%	0.33%	-10.27%	+28.8%	+76.2%
M3	1.03%	0.33%	-68.00%	0.22%	0.15%	-28.84%	+37.8%	+15.2%
b19	1.50%	0.61%	-59.27%	0.37%	0.34%	-7.56%	+25.1%	+84.1%
Avg.	1.20%	0.61%	-56.07%	0.32%	0.31%	-6.42%	+24.4%	+68.9%

Table 6.3: Comparison of CAT vs. WF-CAT ATPG results for twelve circuits w.r.t static-fault coverages *fc* and *wfc*, chip-level test pattern count, and ATPG compute time.

basis on WF-DDMs. The results show that on average, WF-CAT reduces the weakest-fault coverage escapes ($1 - wfc$) with 56.1%. Column 'Hard FCov. Escapes (= $1 - fc$)' shows the comparison of the hard fault coverage escapes between regular CAT and WF-CAT. The results show that our WF-CAT achieves a higher fault coverage also on hard faults.

The last two columns present two other key performance indicators for ATPG: test execution time and test generation time. In column 'Δchip patterns', we give the increase in test patterns required by WF-CAT in comparison to regular CAT. Similarly, the last column shows the increase of the ATPG compute time(including ATPG, fault simulation, and top-off ATPG) for WF-CAT. Both columns show that on average, to achieve 56.1% reduction on the weakest-fault coverage escapes, we need 24.4% more test patterns (i.e., more test execution time) and 68.9% more ATPG compute time (i.e., test generation time) per chip design.

The dynamic ATPG based on two-cycle cell pattern detections shows that on average we increase the fault coverage from 74.8% to 75.5%. The absolute increase is similar to the static fault coverage increase. However, due to the difficulty of two-cycle pattern expansion, the weakest-fault coverage is low, and therefore, the *wfc* reduction (= 3.28%) is relatively low. The cost is on average 20.5% more chip-level patterns and 66.3% more ATPG compute time. The hardest-fault coverage increases from 80.5% to 80.8%, which constitutes a 1.68% *fc* reduction.

6.6 Conclusion

CAT based on only one resistance value for short defects and another one for opens compromises the resulting test quality. If the considered defect resistance value is hard (weak), for a significant fraction of the potential defects this leads to incorrect, oversized (undersized) cell-pattern sets for weaker (harder) variants of those defects, and hence to overestimation (underachievement) of the actual fault coverage.

This chapter presents WF-CAT, which targets per defect location the weakest still-detectable defect variant, as its detection implicitly guarantees detection of all harder variants of the same defect, as well as a 'super-hard' variant of the defect. We have built an experimental tool flow on top of Cadence's Modus CAT functions, consisting of two steps. Step 1 characterizes library cells, by first identifying their potential intra-cell short and open defect locations. Next, analog simulation for a set of defect resistance values determines the critical (weakest still-detectable) resistance per defect-pattern combination, which is stored in a CR(B)M. Step 2, cell-aware ATPG, uses the CR(B)M to target both the weakest and 'super-hard' faults, thus covering both ends of the detectable-defect resistance range. Defect characterization is time consuming, and even more so if performed for multiple resistances values. If defects are simulated from hard to weak, we can reduce simulation time significantly by dropping defects once they become undetectable. Experimental results for a set of eleven benchmark circuits show that on average WF-CAT reduces static *wfc* escapes with 56.1% at a cost of 24.4% more chip-level test patterns and 68.9% more ATPG compute time; and reduces dynamic *wfc* escapes with 3.28% at a cost of 20.5% more chip-level test patterns and 66.3% more ATPG compute time.

CHAPTER 7

Application of Cell-Aware Test on an Advanced 3nm Technology Node

The traditional scaling strategy, based on decreasing *minimum metal pitch* (MMP) and *contacted poly pitch* (CPP) alone, does no longer provide sufficient performance benefits for advanced nodes below 10nm [Bardon et al., 2016]. Hence, designers of standard-cell libraries have now also become directly involved in the scaling rat race. They implement innovative library-cell design concepts (referred to as *scaling boosters*) that provide additional area, performance, and power benefits. The combined benefits of pitch scaling and cell-design innovation obtain chip area reduction of $\sim$40%, which allows us to (more or less) maintain Moore's Law [Schaller, 1997], but more importantly to justify the financial investments required to transition to the next technology node [Sherazi et al., 2019]. However, the design innovations cause standard-cell structures in the advanced nodes to be more complex, and hence make the standard cells more susceptible to manufacturing defects than their predecessors. CAT explicitly targets cell-internal resistive open and short defects to reduce the test escapes compared with conventional ATPG approaches. In this chapter, we report the application of CAT for sub-10nm technology nodes. As a vehicle we use IMEC's iN5 standard-cell library (which corresponds to what commercial foundries would refer to as a '3nm' CMOS node). To the best of

our knowledge, we are the first to report CAT results for a sub-10nm standard-cell library. The results demonstrate that, despite the increased complexity in the library-cell architecture, (1) our CAT tool flow and all optimization work on this tool flow function correctly and (2) the number of potential open- and short-defects that need to be simulated for characterization and later to be covered by cell-aware ATPG is very similar to mature technologies.

The remainder of this chapter is organized as follows. We describe several of the technology and cell-design innovations that are encountered in advanced sub-10nm nodes and detail IMEC's iN5 technology node and corresponding standard-cell library in Section 7.1. CAT library characterization results are presented in Section 7.2. Section 7.3 compares the library characterization results on comparable cells between the iN5 library and Cadence's GDPK045 library in 45nm CMOS. In Section 7.4, we apply the ATPG optimization work presented in Chapters 3 and 5 on twelve benchmarks mapped onto the iN5 library and show the results. Section 7.5 summarizes this chapter.

7.1 iN5 Standard-Cell Library

Research institute IMEC explores the various technology-node scaling directions for and with the semiconductor industry. Engineering teams from IMEC and its partners' couple theoretical research to practical implementation in IMEC's state-of-the-art IC manufacturing facilities. In that context, IMEC has developed experimental process technology nodes and corresponding standard-cell libraries that are named iN8, iN7, and iN5 and correspond to what commercial foundries would refer to as respectively '7nm', '5nm', and '3nm' CMOS nodes. In this section, we describe the scaling path to and several technology and design features of the most advanced of these nodes, viz. iN5. Feasibility of the iN5 technology and standard-cell library has been demonstrated in a joint IMEC-Cadence test chip of a 64-bit microprocessor, for which the tape-out was reported in 2018 [Merritt, 2018].

Library cells are pre-designed and pre-layouted blocks that implement a small logic function. The complexity ranges from basic cells such as buffers, inverters, logic AND, and OR functions to more complex cells such as AND-OR-INVERTER combinations, multiplexers, adders, and scan flip-flops. A cell library typically offers a plethora of functions, and for each function a variety of drive strengths. Standard cells are designed with a fixed layout height, and identical power and ground rails running through the top resp. the bottom of each cell, such that a large logic function can be laid-out as a number of rows of simply concatenated standard cells.

Traditionally, the transition from one technology node to the next was effectuated by

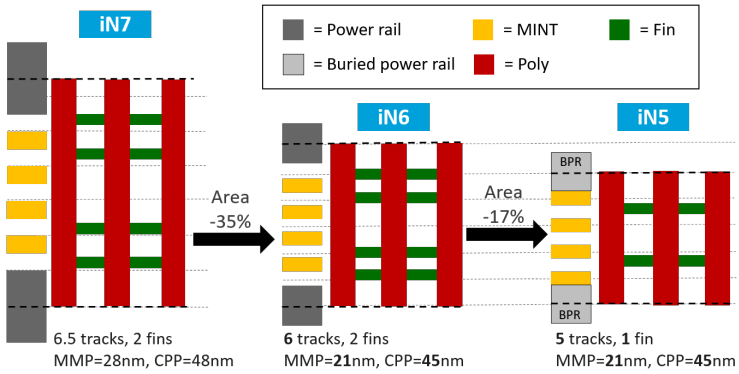
scaling both dimensions of all layout items with a factor 0.7, which resulted in $0.7 \times 0.7 = 49\%$ of the original area for the same function. Many changes have been implemented in subsequent technology nodes, leading up to today's 10nm node. Planar transistors have been replaced by FinFETs since the 22nm node. For sub-10nm nodes, various novel devices, such as nanosheet, forksheet, and CFET [Ryckaert et al., 2019, Weckx et al., 2019] were introduced. In mature nodes, only one metal layer, M1, was used for cell-internal interconnections and power rails. Metal wires could be routed with 90°-degree bends. However, in sub-10nm nodes, for patterning reasons [Vaidyanathan et al., 2012, Ryckaert et al., 2015], such bends are no longer allowed, and metal wires can run exclusively either in the horizontal or vertical direction only. To support routability within cells, an extra intra-cell metal layer named MINT has been introduced: MINT runs in the horizontal direction and M1 in the vertical direction.

The area of a library cell is determined by the product of its height and its width. For such advanced-node libraries, the height is the product of MMP and the number of *tracks*; for standard cells, this implies that both MMP and track count are fixed throughout the entire library. The width of the standard cells is the product of CPP and the number of polylines; the latter is determined by the complexity of the logic function that the cell implements.

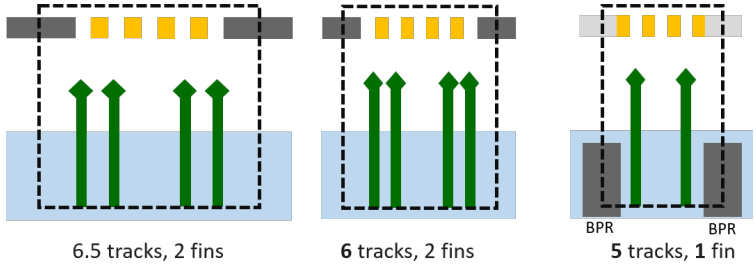
For sub-10nm nodes, scaling MMP and CPP only does not bring sufficient performance benefit anymore. IMEC adopts the *design/technology co-optimization* (DTCO) concept to involve library cell designers to maintain sufficient area, performance, and power benefits for scaling below 10nm [Ryckaert et al., 2014, Ryckaert et al., 2015]. An effective strategy for some of the node transitions is to reduce the number of metal tracks (MINT, in this case), in addition to pitch scaling [Sherazi et al., 2016]. Track scaling was employed in the definition of IMEC's 7nm, 5nm, and 3nm technology nodes.

Figure 7.1 shows in conceptual drawings the scaling steps from IMEC's iN7 ('5nm') to iN5 ('3nm') FinFET nodes, with iN6 as an intermediate ('half') node [Sherazi et al., 2019]. In the step from iN7 to iN6, MMP is scaled with 25% from 28nm down to 21nm. CPP scaling is only a meagre 6.25% from 48nm down to 45nm, as the overall industry trend is that poly pitch scaling slows down due to difficulties to keep pace with device performance and yield issues. Even though from the 45nm technology onwards, the gate material was changed from polysilicon to metal, the term 'poly' is still kept as gate, hence CPP is metal gate pitch for iN5 technology. Along with scaling the number of metal tracks from 6.5 down to six, a total area reduction of 35% is achieved. In order to reach the iN5 node, maintaining six metal tracks is not feasible [Sherazi et al., 2019], and therefore, the iN5 area benefits are due to an additional track reduction down to five tracks. However, with two metal tracks dedicated to power and ground rails, the three remaining metal tracks would be too few to allow sufficient routability freedom inside the cells. Therefore, it was decided to move the power rails to a level below the

device layer; so-called *buried* power rails (BPRs) [Mocuta et al., 2018], and this frees up the necessary space in a five-track standard cell height for four metal tracks as cell-internal routing resources. As Figure 7.1 shows, the five-track cell height also forced us to drop one of the two fins per FinFET transistor. From iN6 to iN5, an additional 17% area reduction is achieved, making the area benefits step from node iN7 to iN5 $(1 - 0.35) \times (1 - 0.17)$=54%, well above the required minimum scaling target of 40%. Under the condition of scaled area, an alternative device, nanosheet FET, is designed for iN5 technology.



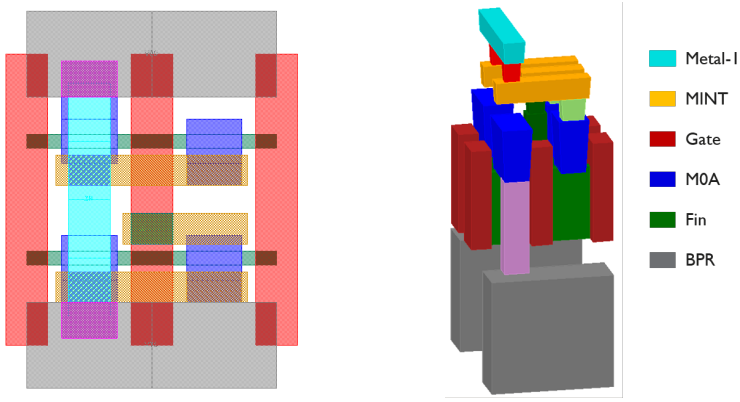
(a) Layout.



(b) Cross section.

Figure 7.1: Scaling from iN7 (= '5nm'), via half-node iN6, to iN5 (='3nm').

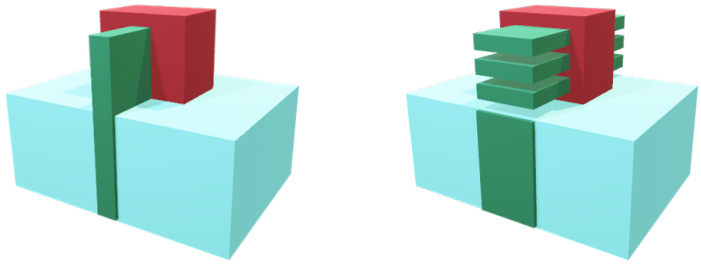
Figure 7.2 shows as example the inverter cell INVD1 from the iN5 library. Figure 7.2(a) shows the top-view layout and Figure 7.2(b) depicts a three-dimensional model of the cell. In the iN5 technology, the layer sequence, from bottom to top, is BPR, the device layer, MINT, and Metal-1. The device layer contains fin, metal gate, and M0A. M0A provides contacts to fins, acting as source or drain of the FinFET transistors.



(a) Layout. (b) 3D model.

Figure 7.2: iN5 library cell INVD1.

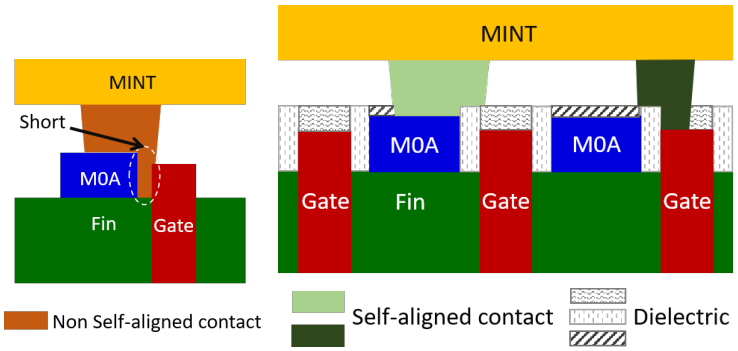
The reduction to a single fin per transistor brings some problems. First, to achieve sufficient performance benefit we have to increase the fin height along z axis dimension, which brings a challenge for processing. Moreover, process variability in a single fin device will heavily affect the device electrical behavior. As a result, a nanosheet device has been introduced to address these issues. [Ryckaert et al., 2019] shows that a 5-track nanosheet provides superior performance to a 5-track single fin inverter. However, new process challenges are also brought on by the nanosheet device. The structures of FinFET and nanosheet devices are shown in Figure 7.3.



(a) FinFET. (b) Nanosheet.

Figure 7.3: Device structure.

New techniques are applied on the contacts between layers. Due to the scaled CPP, the space between source or drain and gate is quite narrow. Hence, shorts are likely to occur if a contact to either of them is misaligned, as shown in Figure 7.4(a). To prevent this, self-aligned contacts are introduced, as depicted in Figure 7.4(b). Dielectrics are placed between and on gates and M0As. In case of some misalignment, the dielectrics prevent contacts to form shorts between gate and source or drain, but of course negatively affects the contact area. Self-aligned contacts help to reduce CPP even further.

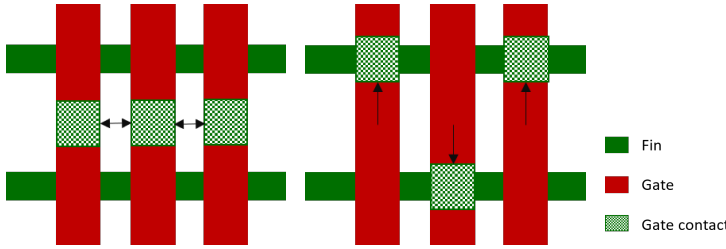


(a) Non self-aligned contact. (b) Self-aligned contact.

Figure 7.4: Preventing contacts from shorting gate and source/drain.

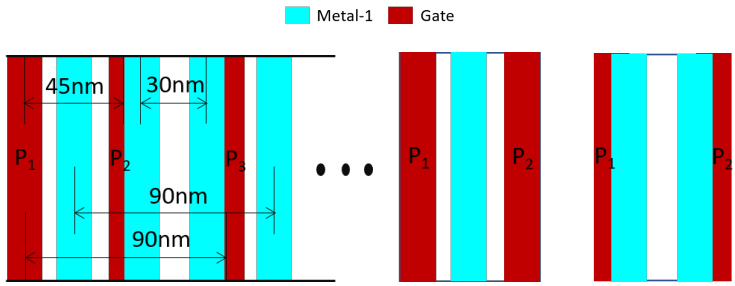
The scaled CPP also impacts gate-contact locations. In Figure 7.5(a), the narrow CPP leads to high parasitic capacitances between contacts, indicating a high likelihood for short defects. Furthermore, as all gate contacts go to the MINT layer and all MINT interconnects only run in the horizontal direction, it is impossible to arrange the interconnections on the MINT layer for these contacts on one line. A solution is shown in Figure 7.5(b). Gate contacts are shifted up and down for two neighboring gate bars, respectively. As the fin count is reduced to one, the contact area on M0A depends on the overlay area between MINT wire and M0A. Due to patterning limitations, we use a single contact on the maximum overlay area. In a planar technology, parallel contacts from metal to a diffusion well are used to increase manufacturing yield.

In the iN5 technology, CPP = 45nm and Metal-1 pitch = 30nm; this is a ratio of 3:2. This implies that between every two gate pitches, we can arrange three Metal-1 pitches, or, in other words, in the area of three gate lines, we can at most fit four Metal-1 lines; see Figure 7.6(a). Consequently, within every two gate pitches, there are two versions of Metal-1 routing possible, as shown in Figures 7.6(b) and 7.6(c), respectively. Metal-1 routing starts from either the middle of two gates or one pitch left-shifted. If a first cell ends up with its relative location between gate and Metal-1, like in Figure 7.6(b),



(a) Gate contacts in middle. (b) Gate contacts moved.

Figure 7.5: Moving gate contacts from (a) the center to (b) the extremes of the gate lines.



(a) Poly-to-M1 pitch ratios. (b) Version A (c) Version B.

Figure 7.6: Three Metal-1 pitches and two gate pitches between every 90nm leads to two versions of Metal-1 routing between every two gate lines.

a cell placed on the right-hand side of this first cell should start gate and Metal-1 like Figure 7.6(c) to prevent Metal-1 being off-grid.

7.2 iN5 Library Characterization

In this section, we first present the defect locations in the iN5 cells identified by the approach in Chapter 3. Next, defect characterization results are presented. Modus v19.1 performs CAT by invoking Quantus v18.1 and Spectre v18.1 for defect-location identification and defect characterization, respectively.

The iN5 standard-cell library contains 116 logic cells. 58 cells differ in logic function and/or drive strength; each of them has two versions of the Metal-1 wire locations. Typ-

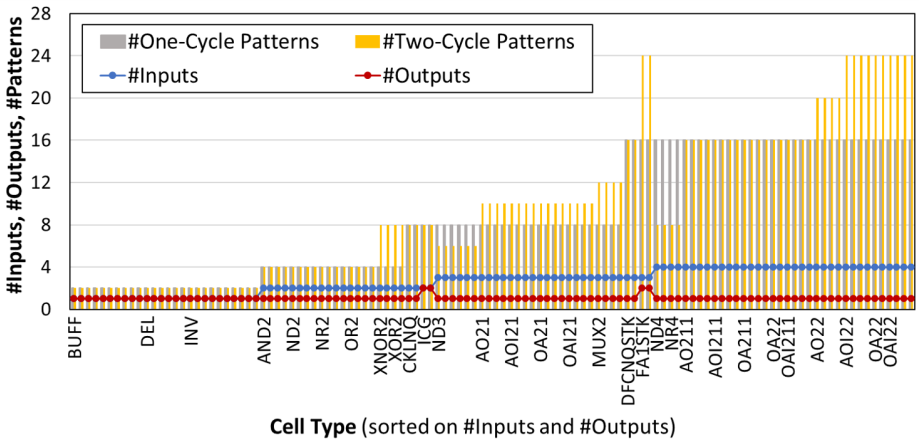


Figure 7.7: The number of inputs, outputs, and patterns of the iN5 library cells.

ically, a library cell with base drive strength denoted by 'X1' or 'D1' in the cell name contains only the basic number of transistors that implements the cell's logic function. We refer to these transistors as *function transistors*. Designers increase the drive strength for cells by adding so-called *drive transistors* in parallel with the function transistors. All these parallel transistors' gates are controlled by the same input signal so that all these transistors can conduct or disconnect the same net pair at the same time, hence the drive strength is increased. In Figure 7.7, we show cell input count n, output count m, the exhaustive one-cycle fully-specified pattern count $m \times 2^n$ (every pattern contains a n-bit fully-specified stimulus pattern and a single-bit response), and the number of two-cycle fully-specified patterns with only one input bit change causing at least one output change from the first to the second cycle. The reasons for selecting such two-cycle cell patterns are (1) two-cycle patterns aim to detect delay faults. Only when a transition occurs at the output, can we measure the delay and then determine the delay faults; (2) two-cycle patterns with multiple-input changes require the multiple-input transitions arriving at the cell inputs simultaneously from chip inputs during chip testing. This requirement adds an extremely strict timing constraint for the ATPG justification, and therefore two-cycle patterns with multiple-input changes are rarely achieved during ATPG. Even though we spend a lot of simulation time with these patterns, they cannot be expanded successfully during ATPG. Considering the trade-off between test quality and cost, we exclude the two-cycle patterns with multiple-input changes. This brings the simulation time under control. Figure 7.7 shows that the difference between the number of one- and two-cycle cell patterns is small.

For the 116 cells, we identified a total of 12,727 full-set open-defect locations; i.e. an

average of 109.7 open-defect locations per cell, shown in Figure 7.8. However, to reduce the simulation time, we first reduce the full set to a compact set based on fault equivalence. Only 47.9% representative opens are selected into the compact set. Many opens on the intra-cell interconnections are equivalent to transistor terminal opens. Only 1,531 out of 3,835 transistor-terminal opens that are not equivalent to any interconnect open are left.

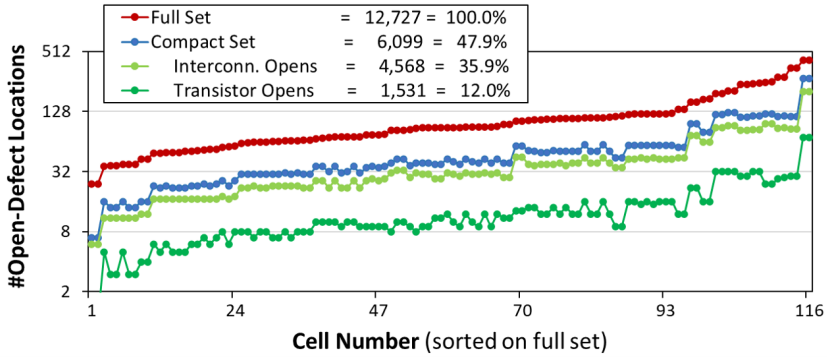


Figure 7.8: Identified open-defect locations in iN5 library cells.

An even larger reduction is obtained for the short-defect locations. The number of short locations in the compact set is reduced by 80.6% from the full set, shown in Figure 7.9. On average, 30.9 shorts are selected into the compact set per cell. Net-pair shorts in the compact set already cover 2,333 out of 2,616 transistor-terminal shorts, and only 283 transistor-terminal shorts that are not equivalent to the net-pair shorts need to be added to the compact set. For opens and shorts combined, the full set of defect locations over all iN5 library cells is reduced in the compact set by 73.8%, leading to a proportional reduction of defect simulation time of the CAT flow.

We consider intra-cell hard open and short defects with resistance values 1000GΩ respectively 0Ω. We perform three rounds of defect simulations as described in Chapter 3. In the first round, the compact set of defects is simulated. Each defect in the compact set maps to a collapsed defect group. Based on the first-round simulation results, we check for each collapsed defect group if all defects are equivalent. For a collapsed defect group that may contain non-equivalent defects, we perform a second simulation to verify. If a collapsed defect group indeed contains several non-equivalent defects, all of the defects in this group are simulated in the third-round simulation to ensure no defect would be missed. Figure 7.10 shows the number of simulations in the first and second round defect simulations. A defect-pattern tuple requires one simulation. The second round simulation determines that only one collapsed short-defect group contains two groups of equivalent short-defects. The cell-pattern set detecting each of the two short-

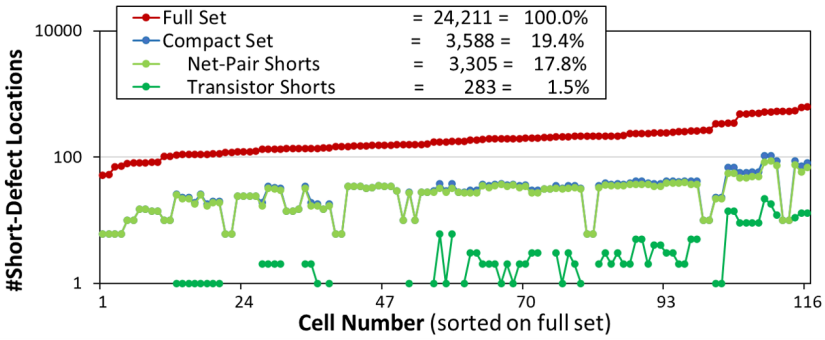
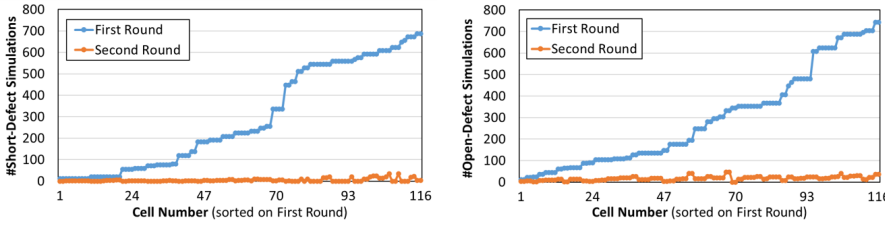


Figure 7.9: Identified short-defect locations in iN5 library cells.

defect groups is also identified by the second-round simulation. Therefore, for both open and short defects, the third-round simulation is omitted. For all 116 cells, the first- and second-round simulation times are 7.1 hours and 19.4 minutes, respectively. By extrapolation, compared to simulating the full set of defects, we save 72.6% simulation time. Applying the cell structure analysis of Chapter 5 to remove the logically undetectable defect-pattern tuples, we reduce the total 7.4 hours simulation time down to 5.2 hours, which constitutes a $(7.4 - 5.2) \,/\, 7.4 = 29.7\%$ reduction.



(a) The number of short-defect simulations.

(b) The number of open-defect simulations.

Figure 7.10: The number of simulations in the first and second round.

Figure 7.11 shows the cumulative defect coverage of one-cycle and two-cycle patterns. The average defect coverage is 72.8%. Low defect coverage happens on cells with high drive strength, like BUFFD16, INVD8, BUFFD8. These cells contain several parallel transistors. An open defect on one transistor terminal will not influence its parallel transistors, and hence does not affect the logic function of the cell. Consequently, they cannot be detected. The two-cycle pattern detection gains the most defect coverage for the two full adder cells circled in the figure.

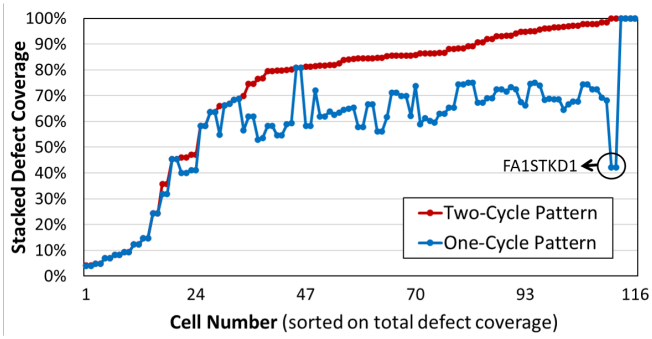


Figure 7.11: Characterization results of the iN5 library.

7.3 GPDK045 Versus iN5

In this section, we compare the standard cells of the iN5 node with a textbook-like, classical technology node. For this purpose, we chose Cadence's GPDK045 45nm CMOS library [Cadence Design Systems, 2014] as a baseline. To enable a fair comparison, we select from both libraries 49 identified cells that have the same logic function and the same relative drive strength. The transistor numbers of iN5 cells are shown in Figure 7.12. The function transistor count increases with the number of cell inputs; they indicate the complexity of the cell design. The number of cell-internal nets always tracks the number of function transistors, as adding parallel drive transistors does not cause an increase in the number of nets. Compared with the 49 iN5 cells in Figure 7.12, 39 out of the 49 GPDK045 counterparts are designed with the same schematic, and hence with the same number of transistors. In the other ten cells, XOR2X1 and XNOR2X1 are implemented in different ways, causing that for both cells the GPDK045 version uses two more transistors. The other eight cells are implemented in the same way, but more transistors in the iN5 version are used to increase the drive strength only, even though the drive strength numbers in the cell names of the two libraries are identical. In total, the 49 selected cells from the iN5 library have 26 transistors and 18 cell-internal nets more than their GPDK045 counterparts.

	Full Set			Compact Set		
	#Opens	#Shorts	#Total	#Opens	#Shorts	#Total
GPDK045	6,121	96,295	102,416 = 100%	2,502	1,555	4,057 = 4.0%
iN5	4,944	9,471	14,415 = 14.1%	2,516	1,543	4,059 = 4.0%
Δ	-1,177	-86,824	-88,001 = -85.9%	+14	-12	+2 = 0.0%

Table 7.1: Comparison of defect-location numbers for GPDK045 and iN5.

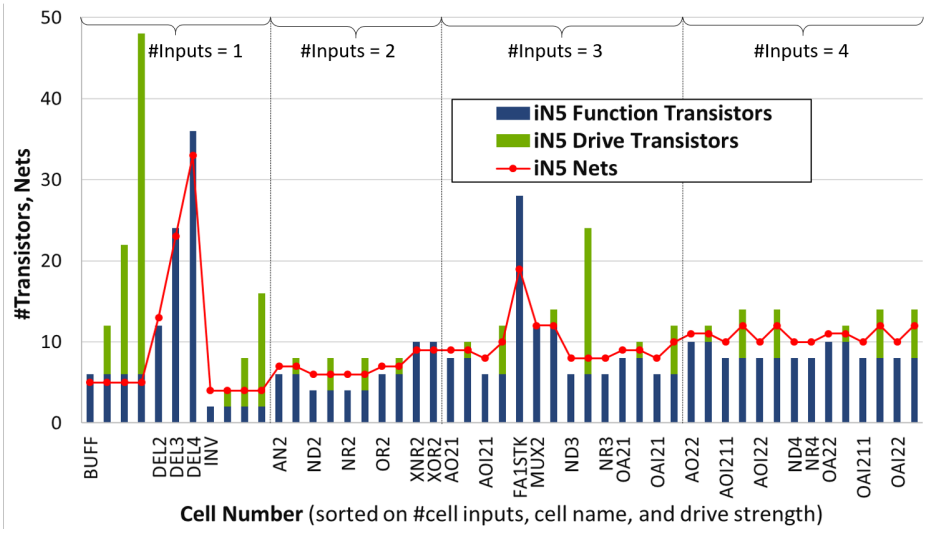


Figure 7.12: The number of transistors and nets of the iN5 cells.

The comparison of the defect-location identification results is shown in Table 7.1. Even though the iN5 cells' structure is more complex than the GPDK045 cells, their full set of defect locations is smaller. This is because, inside library cells, iN5 metal interconnections are in only one direction; the metal wires of the 45nm technology can flexibly change their directions between the x and y axes directions, so that a lot of 90° bend corners exist in the GPDK045 cell layouts. Even though in iN5 cells more interconnection layers and contacts between the layers are designed to compensate for some bend corners, most trivial bend corners are simplified into straight wires. The routing of each interconnection layer in iN5 actually is much simpler than in GPDK045. Bend corners are defect-sensitive points, hence 19.2% more open defect locations are identified based on these bend corners for the GPDK045 cells. Furthermore, the bend corners result in many more parasitic capacitors coupled between their locations and all the other different locations on each net, hence the GPDK045 cells have 90.2% more short locations in their full sets. However, the sizes of the compact sets are quite similar between the two libraries. In total, only two more defect locations are identified for the iN5 library, which causes the difference in defect characterization time to be negligible.

The hard-defect coverage of most iN5 cells is higher than the GPDK045 cells, shown in Figure 7.13. On average, total defect coverages are 61.5% for the GPDK045 and 75.1% for the iN5 library. The non-detectable defects have no effect on cell output even if they are present. Therefore, low defect coverage for a cell means the cell is more robust to defects, which is good news for testing. Higher defect coverage for the iN5 cells implies

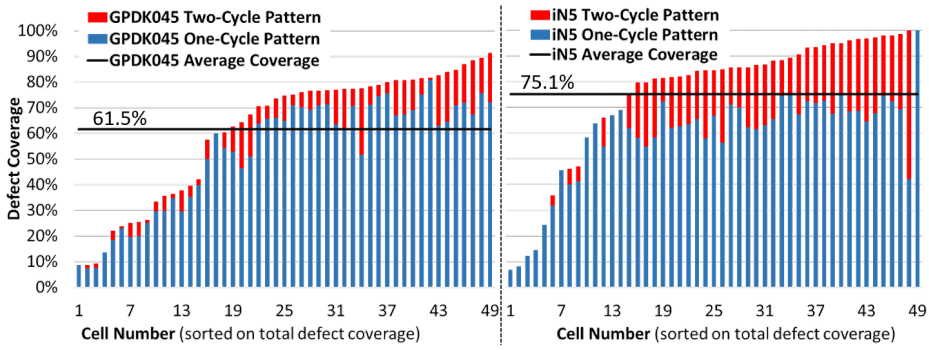


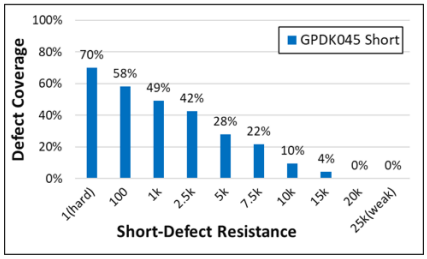
Figure 7.13: Defect coverage of the GPDK045 and the iN5 cells.

that the iN5 cells are more susceptible to hard defects.

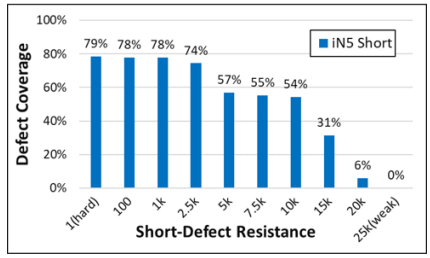
We also compare weak-defect coverages between the two libraries. From hard to weak, we select ten defect resistance values to perform defect simulation. The defect coverages are shown in Figure 7.14. Figure 7.14(a) and Figure 7.14(b) show that short-defect coverage of the iN5 cells is always higher than that of the GPDK045 cells, especially in the short-defect resistance range [7.5kΩ, 20kΩ]. This indicates that the iN5 cells are more susceptible to short defects, especially the weak short defects, than the GPDK045 cells. Similarly, Figure 7.14(c) and Figure 7.14(d) show in the open range [25MΩ, 1MΩ], the iN5 cells have lower defect coverage, which implies that the iN5 cells are more robust to weak open defects than the GPDK045 cells.

7.4 iN5 Cell-Aware ATPG

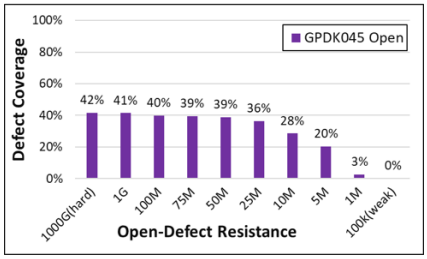
In this section, we show the static-fault ATPG results for eleven circuits (from Cadence, OpenCores, ITC'99 benchmarks, and Arm open cores) mapped onto the iN5 library. We first provide the cell-aware (CA) ATPG results based on original DDMs given by the defect simulation. Afterward, we show the ATPG results optimized by the approach presented in Chapters 4 and 5.



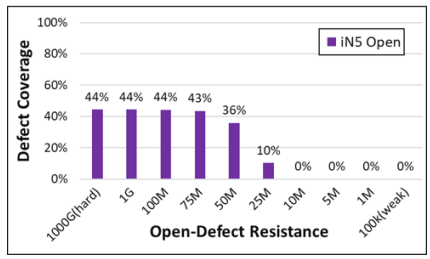
(a) The GPDK045 short-defect coverage.



(b) The iN5 short-defect coverage.



(c) The GPDK045 open-defect coverage.



(d) The iN5 short-defect coverage.

Figure 7.14: From hard to weak, the defect coverages of Libraries GPDK045 and iN5.

7.4.1 ATPG Comparison: Stuck-at vs. Gate-Exhaustive vs. Cell-Aware

To present the CA ATPG KPIs, we compare the CA ATPG results to the results of stuck-at (SA) ATPG and gate-exhaustive (GE) ATPG.

At the cell level, we compare the minimum number of fully-specified cell patterns that cover *all* SA and CA faults, respectively, and also compare them with the number of GE patterns. Determining the minimal number of cell patterns covering all faults per cell is proven to be an $\mathcal{NP}$-hard problem for which we proposed an innovative heuristic algorithm in Chapter 3. From all the 116 iN5 cells which have 988 GE patterns in total, we identify 403 SA patterns and 445 (= 45.0%) CA patterns as a minimum to cover all stuck-at and cell-internal faults, respectively. These selected patterns are referred to as MINCOVER patterns. Figure 7.15 shows the comparison of cell-pattern count per cell. For most cells, the minimum number of SA patterns is equal to the number of CAT patterns. For all sequential cells and for combinational cells full-adder, OAI21, OAI22, OA22, and XNOR2, CAT requires more cell patterns. For cells with few inputs, the minCover CAT cell pattern set is identical to the minCover SA pattern set. Therefore,

for cells with few inputs, CAT cannot show its effectiveness compared to the SA test. For cells with more inputs, the CAT requires unique cell patterns compared to the SA test to explicitly cover the cell-internal defects, and hence CAT covers more faults than the SA test. With the increase of cell-input count, the difference between the number of the CAT and the GE patterns becomes larger, but they achieve the same test quality. Therefore, CAT is more efficient compared with the GE test, especially for cells with a large number of inputs.

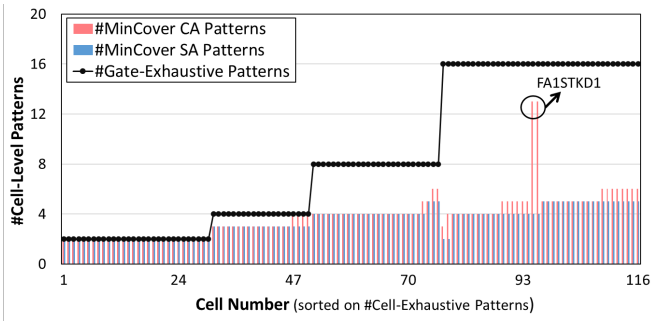


Figure 7.15: The number of cell-level patterns: stuck-at test vs. cell-aware test vs. gate-exhaustive test.

Compared with the GE test, CAT requires many fewer cell-level patterns, hence also fewer chip-level test patterns. For all iN5 cells with 4 (the most) inputs, only 28.5% of cell patterns cover all CA faults. As iN5 is a small library in experimental status, the largest number of cell inputs is only 4; this means that compared to the GE test the test pattern reduction obtained for circuits based on iN5 is rather pessimistic with what would be possible for standard-cell libraries with more complex cells.

The comparison of the number of cell patterns is an indicator of the pattern count comparison at the circuit level. In the following, we show the CAT results at the circuit level. We performed ATPGs with Cadence's tool Modus (version 19.1). In table 7.2, we sort the circuits on the number of cell instances. The average number of cell instances is 20,790. To fairly compare the fault coverage, we perform fault simulations using the three pattern sets generated by CA, SA, and GE ATPG, respectively, against the CA faults to get three fault coverages on the CA faults. The first fault coverage column 'SA FC' shows the SA fault coverage by the SA ATPG. We use the generated SA patterns to perform fault simulation against the CA faults in each circuit. The covered CA fault percentage per circuit is listed in the second fault coverage column 'CA FC for SA patterns'. The SA fault coverage is always higher than the CA fault coverage by the SA test patterns, because, in the SA test, the total number of faults is smaller than that considered by CAT. Figure 7.16(a) shows the number of faults considered by the SA test and

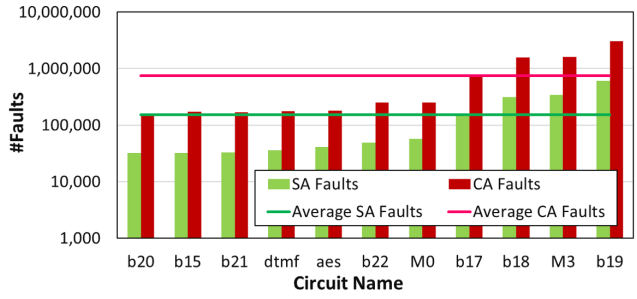
CAT. Please note that the y axis is in logarithmic scale. On average, the number of CA faults is 4.9× SA faults.

Design Data		**Fault Coverage**(FC)				
Name	#Instances	SA FC	CA FC for SA patterns	CA FC for CA patterns	GE FC	CA FC for GE patterns
b20	4,475	99.97%	96.54%	99.22%	84.03%	99.22%
b15	4,508	99.69%	97.61%	98.32%	78.10%	98.32%
b21	4,591	99.94%	96.20%	99.19%	84.28%	99.19%
dtmf	5,054	98.99%	97.68%	99.31%	85.57%	99.32%
aes	5,473	99.99%	98.24%	99.32%	79.96%	99.32%
b22	6,787	99.96%	96.55%	99.20%	84.22%	99.20%
M0	7,603	99.95%	99.42%	99.77%	82.91%	99.79%
b17	19,288	99.46%	96.53%	97.24%	72.02%	97.32%
b18	41,871	99.57%	96.88%	97.50%	73.39%	97.51%
M3	46,124	98.55%	98.43%	99.33%	81.25%	99.36%
b19	82,920	99.55%	96.89%	97.48%	73.34%	97.48%
Avg.	20,790	99.60%	97.36%	98.72%	79.92%	98.73%

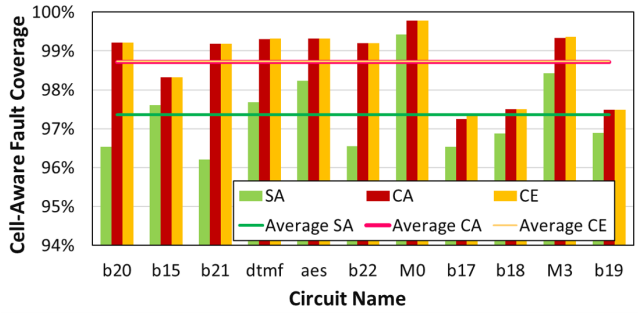
Table 7.2: The design data and fault coverages for eleven circuits. Circuit dtmf is from Cadence [Santosh Malagi, 2019]. Aes is from OpenCores [OpenCores.org, 1999]. M0 and M3 are from Arm [Arm, 2007]. All the other circuits are from ITC99 benchmarks [Scott Davidson, 1999].

By comparing the second and the third fault coverage columns that give the CA fault coverage for the SA test patterns and for the CAT patterns, respectively, we see that most CA faults are covered by the SA test patterns, also shown in Figure 7.16(b). This is because that for each cell instance, a SA cell pattern can cover multiple CA faults. The average CA fault coverage by the SA test patterns is 97.36% compared with 98.74% achieved by the CAT patterns, namely the SA test misses 1.38% CA faults. In the other words, CAT reduces the fault coverage escape from $1 - 97.36\% = 2.64\%$ to $1 - 98.74\% = 1.26\%$. This constitutes a reduction of 52.3% which indicates a significant test escape rate that is quantified in parts per million (PPM). With a better test quality than the SA test, CAT requires 36.6% more test patterns which are considered acceptable. We also perform a top-off CA ATPG on the SA test to generate test patterns for the missed 1.38% CA faults. 46.6% more test patterns in addition to the SA test patterns are generated, but the CA fault coverage is the same with the scratch CA ATPG. This implies that the scratch CA ATPG is more efficient than performing SA ATPG and top-off CA ATPG in sequence.

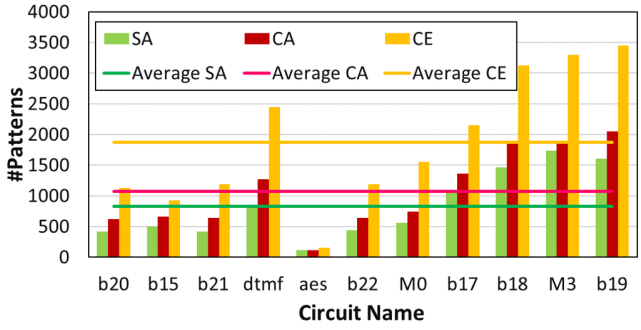
For the eleven circuits, the average fault coverage of the GE test is 79.9% which indicates that only 79.9% cell patterns are expanded successfully per circuit due to the circuit environment. Hapke et al. [Hapke et al., 2011b] consider the percentage of the successfully expanded cell patterns as the GE fault coverage. We do not agree with this, because no faults are considered by the GE test. The faults in the diagonal DDMs are dummy ones to guide the ATPG to expand all cell patterns. To compare the test quality between



(a) Fault count.



(b) Cell-aware fault coverage.



(c) Chip-level pattern count.

Figure 7.16: Stuck-at test vs. cell-aware test vs. cell-exhaustive test.

CA and GE tests, we use the same fault model, i.e., cell-aware fault model. Hence we perform the fault simulation with the GE test patterns on the cell-aware faults per circuit. The results show that the GE test patterns cover the same number of cell-aware faults

as the CAT patterns. However, on average GE test requires 1.7× the number of CAT patterns per circuit. The test pattern count indicates the time spent on an automatic test equipment (ATE) for every under-test-chip, hence CA is much cheaper than the GE test.

In summary, CAT achieves higher test quality with affordable test cost compared with SA test, and achieves the same test quality as GE test with a much lower test cost.

7.4.2 CA ATPG Optimization

We apply the two DDM manipulation algorithms introduced in Chapter 4 on the 116 iN5 cells. Based on the conclusion of Chapter 3, we use the two algorithms in conjunction achieve the best ATPG results. Therefore, we first extend the DDMs by identifying all possible partly-specified cell patterns and then determine the preferential cell patterns with the full fault coverage and the minimum sum of care bits. The results are shown in Figure 7.17 and Table 7.3. The number of identified partly-specified patterns is 1.5× the number of fully-specified patterns. In the extended DDMs, only 1256 (= 18%) cell patterns which contain 31 don't-care bits are required to cover all CA faults.

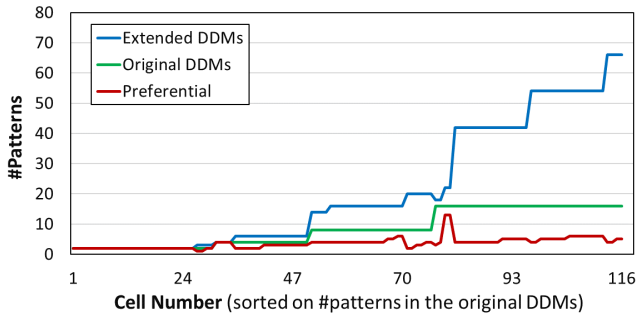


Figure 7.17: The number of cell-level patterns in the original and the extended DDMs and the number of preferential patterns.

	#Patterns		#Care Bits		#Don't-Care Bits
Extended DDM	2488	= 100%	6918	= 100%	2010
Original DDM	988	= 25.2%	3318	= 48.0%	0
Preferential	445	= 17.9%	1256	= 18.1%	31

Table 7.3: DDM manipulation results.

The ATPG results are presented in Table 7.4. This table has two main columns: design data and Δ ATPG results (comparing ATPG with optimized to non-optimized DDMs).

More cell instances with more inputs per cell improve the effectiveness of our optimization algorithms. The column 'Design Data' consists of three sub-columns: the circuit name, the number of cell instances in the circuit $|I|$, and the weighted average number of inputs per cell instance $\sum_{k\in[1..4]} \big(k \times |\{i \in I | n_i = k\}|\big)/|I|$.

Design Data			**Δ ATPG Results**		
Name	$\|I\|$	WAvg. Inp.	FC escapes	Patterns	ATPG Time
b20	4,475	1.17	0%	-12.3%	-20.0%
b15	4,508	2.49	0%	-3.9%	+38.2%
b21	4,591	2.33	0%	-9.7%	+0.5%
dtmf	5,054	2.39	-1.48%	-18.1%	+25.9%
aes	5,473	2.40	0%	+10.5%	+16.2%
b22	6,787	2.33	-0.05%	-7.8%	+4.9%
M0	7,603	2.53	-5.03%	-15.2%	+0.5%
b17	19,288	2.57	-4.90%	+31.8%	+11.3%
b18	41,871	2.56	-0.56%	+12.5%	+56.5%
M3	46,124	2.52	-11.53%	+13.3%	-4.1%
b19	82,920	2.56	-0.11%	+10.7%	+63.6%
Avg.	20,790	2.35	-2.18%	+1.06%	+17.59%

Table 7.4: The optimized ATPG results for eleven circuits.

In the second main column, we compare CA ATPG based on optimized DDMs to non-optimized DDMs. The three sub-columns address the three key performance indicators of ATPG: test quality, test execution time, and test generation time. Test quality is expressed as the fraction of the fault coverage (FC) escapes, i.e., 100% - fault coverage. We report how much our optimization reduces the fraction of non-covered faults. For example: for benchmark circuit $M3$, the non-optimized cell-aware fault coverage was 99.33%, while the optimized fault coverage increased to 99.41%; this constitutes a reduction of the fraction of FC escapes with $(99.41\% - 99.33\%)/(100\% - 99.33\%) = 11.9\%$. This column shows that our approach always has benefits for test quality. On average, we increase the fault coverage from 98.720% to 98.748%, equivalent to a 2.18% reduction of FC escapes for the eleven circuits. Even though 2.18% fault coverage escape is a small reduction, after we convert it to test escape rate, namely defective-part-per-million (DPPM), it becomes a big number. For example, assuming the yield is 90%, we need to filter out the 0.1 million defective circuits per million manufactured circuits. Without using the optimized DDMs, the original number of test escapes is 0.1 *million* $\times\ (1 - 0.9872) = 1280$, and hence the test escape rate is $1280/(0.9 + 0.00128) = 1420$ DPPM. Similarly, with the 98.748% fault coverage, the optimized test escape rate is 1389 DPPM. The test escape reduction is 31 DPPM. Please note, this number is inversely proportional to the yield. If the yield is 80%, the absolute test escape reduction is 70 DPPM.

In column '*Patterns*' we give the delta percentage of the number of test patterns generated with optimized DDMs in comparison to non-optimized ATPG. In the iN5 circuits,

most cells are simple cells with only one or two inputs. Most cell-internal defect detections require all inputs to be specified, and few don't-care bits are identified for these cells. Also, we run two-stage ATPG which is not effective for chip pattern compression. Therefore, the chip-level pattern count is not always reduced. On average, the number of chip patterns increases by 1%. The benefit on fault coverage is achieved through a significantly extended DDM (on average for the eleven circuits 2.5× more cell patterns), and therefore the ATPG execution times increase (on average +17.6%).

7.4.3 Weakest-Fault Coverage Optimization

Per defect, we run for ten defect resistance values respectively from hard to weak. For short defects, we set the defect resistance range to [1Ω, 25kΩ). 1Ω is selected to model a hard short defect, and through simulation, we found that no short defect ≥25kΩ was detectable in any of the library cells. Similarly, for open defects, we set the defect resistance range to [1TΩ, 100kΩ). For either defect resistance range, we select ten resistance values to cut into ten resistance bins; the selected values are shown in Table 7.5.

	Hard									**Weak**
Iteration i	1	2	3	4	5	6	7	8	9	10
$R[\text{short}, i]$ (Ω)	1	100	1k	2.5k	5k	7.5k	10k	15k	20k	25k
$R[\text{open}, i]$ (Ω)	1T	1G	100M	75M	50M	25M	10M	5M	1M	100k

Table 7.5: For short and open defects, ten iterations of defect simulation based on the ten resistance values individually.

Design Data	ΔWeakest-FC escapes	ΔHard-FC Escapes	ΔChip Patterns	ΔATPG Time
b20	-18.8%	0%	+2.6%	+7.8%
b15	-18.2%	0%	+20.3%	+54.3%
b21	-25.7%	0%	+7.4%	+12.8%
dtmf	-21.8%	-1.9%	+9.7%	+25.9%
aes	-14.8%	0%	+7.9%	+13.8%
b22	-20.3%	-0.1%	+7.0%	+32.9%
M0	-31.5%	-5.7%	+8.6%	+31.6%
b17	-11.9%	-5.0%	+41.5%	+38.5%
b18	-9.8%	-0.7%	+30.9%	+56.1%
M3	-23.8%	-9.9%	+22.7%	+14.5%
b19	-9.7%	-0.2%	+29.6%	+83.9%
Avg.	-18.8%	-2.1%	+17.1%	+33.8%

Table 7.6: The weakest-fault ATPG results for eleven circuits.

Based on the defect simulation results, we identify the weakest detectable faults and their cell patterns following the approach in Chapter 5. Afterward, we perform the two-stage weakest-fault ATPG with the WF-CAT flow described in Chapter 5.

The ATPG results are shown in Table 7.6. Columns 'Weakest-FC Escapes' and 'Hard-FC Escapes (= $1 - fc$)' present fault coverage escape data for the weakest faults and the hard faults. These two columns address the main key performance indicator for ATPG: test quality. The larger the reported reduction, the more effective our WF-CAT methodology. The results show that on average, WF-CAT reduces the weakest-fault coverage escapes with 18.8% and reduces the hard-fault coverage escapes with 2.1%. In comparison to regular CAT, the last two columns present two other key performance indicators of ATPG: test execution time and test generation time. In column 'Δchip patterns', we give the increase in the number of test patterns required by WF-CAT in comparison to regular CAT. The costs for achieving the better test quality are on average, 17.1% more test patterns (i.e., more test execution time) and 33.8% more ATPG compute time (i.e., test generation time) per chip design.

7.5 Conclusion

This chapter is, to the best of our knowledge, the first to report on the application of CAT defect characterization on an advanced sub-10nm cell library.

In this chapter, we used IMEC's iN5 library, a standard-cell library in (what commercial foundries would refer to as) a 3nm CMOS technology. The technology uses advanced features, including FinFETs (with the number of fins per transistor reduced to one due to the small dimensions), an intermediate metal layer MINT, buried power rails, and metal wires per layer running in only one direction (i.e., either x or y axis direction). While still experimental in status, the iN5 library, which so far only contains 116 cells, has been used to design and manufacture a 64-bit microprocessor, based on a five-track cell architecture with MMP=21nm and CPP=45nm.

We ran Cadence's CAT library characterization tool flow on the iN5 library and compared the results of the various steps with results obtained on in logic function and drive-strength equivalent cells from the GPDK045 standard-cell library from Cadence [Cadence Design Systems, 2014] in a mature 45nm planar CMOS. Although the iN5 library-cell architecture comprises more layers than GPDK045 cells have, the number of identified non-equivalent potential open- and short-defect locations for both libraries is almost equal. Our DLI algorithms reduced the set of defects to be simulated by 73.8%, thereby reducing the defect simulation time for all 116 combinational library cells to 9.6 hours. The average defect coverage for hard open and short defects is 72.8%. Compared with the GPDK045 technology, the iN5 cells is more robust to weak defects, but more sensitive to hard defects.

Our DDM optimization algorithms found that to cover all detectable defects, we have on average 17.9% *preferential* cell patterns, and the preferential pattern set has 18.1% care bits when compared to the exhaustive set of partially- and fully-specified cell patterns. With the two-stage ATPG for eleven circuits, on average we reduce the hard-fault coverage escape by 2.18% at the costs of 1.0% more patterns and 17.6% more ATPG compute time.

With the weakest-fault ATPG, for eleven circuits on average, we reduce the weakest-fault coverage escape by 18.8% at the costs of 17.1% more test patterns and 33.8% more ATPG compute time.

CHAPTER 8

Conclusion and Future Work

In this chapter, we summarize our contributions to the cell-aware test (CAT) methodology in Section 8.1, and discuss directions for future work in Section 8.2.

8.1 Conclusions

The entire semiconductor industry is pushing to improve the IC test quality, with two main drivers. (1) IC designs integrate an ever growing number of components with ever decreasing sizes, and hence the number of potential defect locations increases. (2) ICs are increasingly used in safety-critical applications (such as automotive, medical, avionics), and these markets demand the highest product quality with zero tolerance for test escapes. An IC typically consists of various blocks, such as digital logic circuits, analog modules, and memories. Different types of blocks require different testing methodologies, respectively. This thesis focuses on logic-circuit testing. CAT, as a new logic test generation approach, is effective to reduce test escapes at an affordable cost. Therefore, several leading semiconductor companies, such as Intel and AMD, already adopted CAT

for the volume testing [Howell et al., 2018, Hapke et al., 2012].

In this thesis, we propose various approaches to improve the CAT methodology with the aim to optimize the three test KPIs in order of decreasing importance: (1) test quality, (2) test execution time, and (3) test development time. With the analysis of the basis of the CAT's effectiveness and efficiency compared to conventional digital test approaches, we use the three elements (cell-internal defects, cell-level test patterns, and detection result per defect-pattern tuple) in a defect detection matrix (DDM) to propose the optimization methodologies. In Chapter 2, we introduced the Cadence CAT tool flow as the baseline for our optimization work. In Chapters 3–6, we present approaches to optimize single or multiple KPIs. In Chapter 7, we apply the CAT tool flows without and with our optimization work on an advanced 3nm technology to demonstrate the universality of our CAT tool flows. We do contribute the following seven innovations to CAT:

1 We are the first to publish the details of the CAT tool flow implementation. In Chapter 2, we disclose how to use the parasitic extraction (PEX) tool to assist the defect-location identification. In Chapter 4, we represent the details of defect simulation results and how to effectively encode and use these results in cell-aware ATPG. The details that are now open to the public have two effects. (1) Cadence's customers using the CAT tool flow can fully understand the methodology behind it. (2) We offer the state-of-the-art of the CAT development to the industry, such that all engineers and researchers in the microelectronics field have an opportunity to make new contributions. This avoids duplicated research due to industrial secrets.
2 We discuss the difference among stuck-at, cell-aware, and gate-exhaustive ATPGs and derive the cause of the CAT effectiveness and efficiency. CAT explicitly targets all possible cell-internal defects, and therefore reduces test escapes compared to stuck-at test, which targets only a subset of defects considered by CAT. However, more cell patterns are required to cover these cell-internal defects. Compared to the gate-exhaustive test, fewer cell patterns are sufficient to cover all defects. Therefore, CAT reduces the number of test patterns and thus test execution time. The information encoded into DDMs, namely defects, cell patterns, and the detection result of each defect-pattern tuple, is the basis for CAT's effectiveness and efficiency. This analysis is the base of the CAT methodology optimization.
3 We prove that selecting the minimum number of cell patterns to cover all cell-internal faults is an $\mathcal{NP}$-hard problem equivalent to the classic Set Covering Problem in the mathematics field and widely encountered in many engineering applications. We propose an innovative approach that outperforms the prior work. Furthermore, we build a two-stage ATPG flow to expand the selected cell patterns first, in order to reduce the test development time.
4 We show that don't-care bits identification for cell patterns brings benefits to both test quality and test execution time, especially for large circuits with more complex cells.

We identify the don't-care bits per defect detection by algorithm instead of by simulation. The compute time is only $\sim$1 minute for 476 cells from Cadence's GPDK045 library, which is negligible. The identified don't-care bits cause the non-covered faults and the number of test patterns to be reduced by 45% and 14%, respectively, on average for twelve benchmark circuits.

5 We observe that many defect-pattern tuples are not detectable. To save the defect simulation time, and hence save test development time, we introduce cell-structure analysis to effectively filter out the logically undetectable defect-pattern tuples before the defect simulation. For 476 cells of Cadence's GPDK045 library, we reduce the simulation time from 1,955 hours ($\sim$ 81 days) to 1,039 hours ($\sim$ 43 days), which constitutes a 47% reduction in the simulation time. For 116 cells of imec's iN5 library, the simulation time is reduced by 29%. The reduction for iN5 library is smaller than for the GPDK045 library, because most iN5 library cells are simple cells with low drive strength, while complex or/and high drive strength cells have much higher percentage of logically undetectable defect-pattern tuples.

6 Both weak and hard defect variants can occur at a defect location. If a weak defect variant is detected by a pattern, all harder defect variants are also detected by this pattern. We propose an approach to detect the maximum detectable interval per defect instead of detecting a single defect variant. After identifying the cell patterns that detect the still-detectable weakest defect variant per defect location, we guide the ATPG tool to expand these cell patterns first to detect the weakest faults caused by the weakest defects as many as possible. The detected weakest faults indicate that we cover the maximum detectable interval at the locations of these faults. This improves the test quality drastically.

7 As far as we know, we are the first to show the applicability of the CAT tool flow and our optimization work on a sub-10nm technology of which cell design is more complex than its predecessors. For this, we use the imec 3nm technology as a vehicle to demonstrate the solutions in this thesis. The results show that the optimized CAT tool flow works on the 3nm complex library cells and the three test KPIs are optimized for the 3nm technology library and circuits, compared to the original CAT tool flow.

8.2 Future Work

Driven by further improving the test quality based on the current CAT methodology, more research needs to be carried out. Specifically, we see promising opportunities on the following three topics:

1 Consider the third fault type, fast fault, for CAT. From the circuit functionality point of view, fault types in logic circuits can be classified into the static fault, delay fault,

and fast fault. A static fault is caused if the signal on a net is always to be opposite to the correct signal, while a delay (fast) fault is caused if a signal transition arrives with an extra positive (negative) delay compared to the defect-free circuit. So far, all testing work is on the static and delay faults. Based on our defect simulation results, some defects cause faster signal transition compared to the defect-free circuit. Therefore, it is necessary to investigate the reasons and impact of the fast faults. If the impact cannot be ignored, we should consider the fast faults also to improve the test quality further.

2 Interweave the MINCOVER algorithm and the ATPG algorithm, with the aim to reduce the number of chip-level test patterns. In this thesis, the proposed MINCOVER algorithm is independent of the cell-aware ATPG. We identify the preferential cell patterns per cell DDM with the MINCOVER algorithm and perform two-stage ATPG to reduce the ATPG compute time. The first stage ATPG expands only the preferential cell patterns and the second stage ATPG expands the remaining cell patterns in each DDM. Typically, in the first stage ATPG, not all preferential cell patterns can be expanded successfully for each cell instance. The defects uniquely detected by the non-expandable cell patterns can not be covered, and therefore the fault coverage at the circuit level is not satisfactory. But perhaps, the non-covered defects in the first stage ATPG can be covered by expanding the non-preferential cell patterns in the second stage ATPG which is used to compensate the fault coverage. However, we do not think that this is the most effective way to use the MINCOVER algorithm. Interweaving the MINCOVER algorithm and the cell expansion (i.e. ATPG) algorithm is better for the following reasons. In a circuit design, the expandable cell patterns are specific to each cell instance because the probability of a successful expansion depends on the surrounding cell-instance connections. For a particular cell instance, if a cell pattern is selected as a preferential pattern, but is not expandable, we should remove this pattern from this cell instance DDM. The removal makes the preferential pattern set based on the remaining DDM different from the preferential pattern set based on the original DDM. Therefore, the MINCOVER algorithm should be applied again for the particular cell instance. Thus, during the ATPG process, the MINCOVER algorithm selects the preferential cell patterns per cell instance, and the ATPG engine expands only the preferential cell patterns. It thus expands the fewest cell patterns in the DDM excluding the non-expandable cell patterns. In theory, interweaving the MINCOVER algorithm and the ATPG algorithm causes the number of expanded cell patterns per cell instance in a circuit to be minimized. Therefore, the chip-level test pattern count would be also minimized. However, we need to optimize the DDM for each cell instance.

3 Device-aware test. The conventional logic test approaches treat every library cell as a black box and consider faults at the cell boundary with the expectation that the cell-boundary faults are the only fault effects caused by cell-internal defects. However, CAT proves that the cell-boundary faults considered by the conventional logic tests are not sufficient to represent all cell-internal defects. CAT opens each library cell and

considers the open defects on and short defects between cell-internal interconnections and transistor terminals. However, this means CAT treats each transistor as a black box with the expectation that the short and open defects at the transistor boundaries would represent all the abnormal electrical behaviors caused by the transistor-internal physical defects. The question is: can the defects assumed at the transistor boundaries model all transistor internal physical defects? Is it necessary to open each transistor to investigate the electrical fault behaviors caused by various transistor-internal defects?

The above-mentioned topics will also bring new trade-off between test quality, test execution time, and test development time. Therefore, after a further research on these topics, tool vendors can offer more user options in the CAT tool flow to meet customers' various requirements on the test KPIs.

Publication List

The list of publications based on our research is as following.

International Conferences

1. **Z. Gao**, S. Malagi, E.J. Marinissen, J. Swenton, J. Huisken, and K. Goossens. “Defect-Location Identification for Cell-Aware Test". *Proceedings of the IEEE Latin-American Test Symposium (LATS)*, pages 1-6, Santiago, Chile, March 2019. doi:10.1109/LATW.2019.8704561. **(Best Paper Award)**

2. **Z. Gao**, M. Hu, J. Swenton, S. Malagi, J. Huisken, K. Goossens, and E.J. Marinissen. “Optimization of Cell-Aware ATPG Results by Manipulating Library Cells’ Defect Detection Matrices". *Proceedings of the IEEE International Test Conference – Asia (ITC-Asia)*, pages 91-96, Tokyo, Japan, September 2019. doi:10.1109/ITC-Asia.2019.00029.

3. **Z. Gao**, M. Hu, R. Baert, B. Chehab, S. Malagi, J. Swenton, J. Huisken, K. Goossens, and E.J. Marinissen. “Application of Cell-Aware Test on an Advanced 3nm CMOS Technology Library". *Proceedings of the IEEE International Test Conference (ITC)*, pages 1-6, Washington, D.C., USA, November 2019. doi:10.1109/ITC44170.2019.9000164.

4. M. Hu, **Z. Gao**, S. Malagi, J. Swenton, J. Huisken, K. Goossens, C. Wu, and E.J.

Marinissen. “Tightening the Mesh Size of the Cell-Aware ATPG Net for Catching All Detectable Weakest Faults". *Proceedings of the IEEE European Test Symposium (ETS)*, pages 1-6, Tallinn, Estonia, May 2020. doi:10.1109/ETS48528.2020.9131567. **(Best Paper Award nomination)**

5. F. Lorenzelli, **Z. Gao**, J. Swenton, S. Malagi, and E.J. Marinissen. “Speeding up Cell-Aware Library Characterization by Preceding Simulation with Structural Analysis". *Proceedings of the IEEE European Test Symposium (ETS)*, pages 1-6, on-line, Belgium, May 2021. doi:10.1109/ETS50041.2021.9465469. **(Best Paper Award nomination)**

Scientific Journal

1. **Z. Gao**, M. Hu, J. Swenton, S. Malagi, J. Huisken, K. Goossens, and E.J. Marinissen. “Reducing Library Characterization Time for Cell-aware Test while Maintaining Test Quality". *Journal of Electronic Testing Theory and Applications (JETTA)*, Springer, volume 37, issue 2, pages 161–189, April 2021. doi:10.1007/s10836-021-05943-3.

Bibliography

[Agathos and Papapetrou, 2013] Agathos, S. and Papapetrou, E. (2013). On the Set Cover Problem for Broadcasting in Wireless Ad Hoc Networks. *IEEE Communications Letters*, 17(11):2192–2195. doi:10.1109/LCOMM.2013.091913.131765.

[Andjelkovic et al., 2018] Andjelkovic, M. et al. (2018). Impact of Resistive Open and Bridge Defects on the SET Robustness of Standard CMOS Combinational Logic. In *Proc. IEEE East-West Design & Test Symposium (EWDTS)*, pages 1–6. doi:10.1109/EWDTS.2018.8524748.

[Arai et al., 2017] Arai, M., Inuyama, S., and Iwasaki, K. (2017). Layout-Aware 2-Step Window-Based Pattern Reordering for Fast Bridge/Open Test Generation. In *Proc. IEEE International Test Conference (ITC)*, pages 1–8. doi:10.1109/TEST.2017.8242060.

[Arm, 2007] Arm (2007). Arm Cortex-M Series Processors. `https://developer.arm.com/ip-products/processors/cortex-m`.

[Arumí et al., 2008] Arumí, D., Rodriguez-Montanes, R., and Figueras, J. (2008). Experimental Characterization of CMOS Interconnect Open Defects. *IEEE Transactions on Computer-Aided Design*, 27(1):123–136. doi:10.1109/TCAD.2007.907255.

[Bardon et al., 2016] Bardon, M. G. et al. (2016). Extreme Scaling Enabled by 5 Tracks Cells: Holistic Design-device Co-optimization for FinFETs and Lateral Nanowires. In *Proc. IEEE International Electron Devices Meeting (IEDM)*, pages 28.2.1–28.2.4. doi:10.1109/IEDM.2016.7838497.

[Blanton et al., 2001] Blanton, R., Dwarakanath, K., and Shah, A. (2001). Analyzing the Effectiveness of Multiple-detect Test Sets. In *Proc. IEEE International Test Conference (ITC)*, pages 876–885. doi:10.1109/TEST.2003.1271073.

[Bushnell and Agrawal, 2000] Bushnell, M. and Agrawal, V. (2000). *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*. Springer, Boston, MA. doi:10.1007/B117406.

[Cadence Design Systems, 1999] Cadence Design Systems (1999). Cadence Standard Parasitic Format (SPF).

[Cadence Design Systems, 2014] Cadence Design Systems (2014). Reference Manual Generic 45nm Salicide 1.0V/1.8V 1P 11M Process Design Kit and Rule Decks (PRD) Revision 4.0.

[Cadence Design Systems, 2018] Cadence Design Systems (2018). Dracula® Reference, product version ic6.1.7.

[Cho et al., 2005] Cho, K., Mitra, S., and McCluskey, E. (2005). Gate Exhaustive Testing. In *Proc. IEEE International Test Conference (ITC)*, pages 1–7. doi:10.1109/TEST.2005.1584040.

[Chvátal, 1979] Chvátal, V. (1979). A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235.

[Dixit et al., 2018] Dixit, S., Vora, D., and Peng, K. (2018). Challenges in Cell-Aware Test. In *Proc. IEEE European Test Symposium (ETS)*, pages 1–6. doi:10.1109/ETS.2018.8400700.

[Eichenberger et al., 2008] Eichenberger, S. et al. (2008). Towards a World Without Test Escapes: The Use of Volume Diagnosis to Improve Test Quality. In *Proc. IEEE International Test Conference (ITC)*, pages 1–10. doi:10.1109/TEST.2008.4700604.

[Eldred, 1959] Eldred, R. (1959). Test Routines Based on Symbolic Logical Statements. *ACM Transactions on Design Automation of Electronic Systems*, 6(1):33–36. doi:10.1145/320954.320957.

[Fantini and Morandi, 1985] Fantini, F. and Morandi, C. (1985). Failure modes and mechanisms for VLSI ICs - a review. *IEE Proceedings G - Electronic Circuits and Systems*, 132(3):74–81. doi:10.1049/ip-g-1.1985.0018.

[Ferguson and Larrabee, 1991] Ferguson, F. and Larrabee, T. (1991). Test Pattern Generation for Realistic Bridge Faults in CMOS ICs. In *Proc. IEEE International Test Conference (ITC)*, pages 492–. doi:10.1109/TEST.1991.519711.

[Ferguson and Shen, 1988a] Ferguson, F. and Shen, J. (1988a). A CMOS Fault Extractor for Inductive Fault Analysis. *IEEE Transactions on Computer-Aided Design*, 7(11):1181–1194. doi:10.1109/43.9188.

[Ferguson and Shen, 1988b] Ferguson, F. and Shen, J. (1988b). Extraction and Simulation of Realistic CMOS Faults using Inductive Fault Analysis. In *Proc. IEEE International Test Conference (ITC)*, pages 475–484. doi:10.1109/TEST.1988.207759.

[Galiay et al., 1980] Galiay, J., Crouzet, Y., and Vergniault, M. (1980). Physical Versus Logical Fault Models MOS LSI Circuits: Impact on Their Testability. *IEEE Transactions on Computers*, C-29(6):527–531. doi:10.1109/TC.1980.1675614.

[Gao et al., 2019a] Gao, Z. et al. (2019a). Application of Cell-Aware Test on an Advanced 3nm CMOS Technology Library. In *Proc. IEEE International Test Conference (ITC)*, pages 1–6. doi:10.1109/ITC44170.2019.9000164.

[Gao et al., 2019b] Gao, Z. et al. (2019b). Defect-Location Identification for Cell-Aware Test. In *Proc. IEEE Latin-American Test Symposium (LATS)*, pages 1–6. doi:10.1109/LATW.2019.8704561. (Best Paper Award).

[Geuzebroek et al., 2007] Geuzebroek, J. et al. (2007). Embedded Multi-Detect ATPG and Its Effect on the Detection of Unmodeled Defects. In *Proc. IEEE International Test Conference (ITC)*, pages 1–10. doi:10.1109/TEST.2007.4437649.

[Gulati and Charles, 1993] Gulati, R. and Charles, F. (1993). *Iddq Testing of VLSI Circuits*. Springer, Boston, MA, USA.

[Hapke et al., 2009] Hapke, F. et al. (2009). Defect-Oriented Cell-Aware ATPG and Fault Simulation for Industrial Cell Libraries and Designs. In *Proc. IEEE International Test Conference (ITC)*, pages 1–10. doi:10.1109/TEST.2009.5355741.

[Hapke et al., 2010] Hapke, F. et al. (2010). Defect-Oriented Cell-Internal Testing. In *Proc. IEEE International Test Conference (ITC)*, pages 1–10. doi:10.1109/TEST.2010.5699229.

[Hapke et al., 2011a] Hapke, F. et al. (2011a). Cell-Aware Analysis for Small-Delay Effects and Production Test Results from Different Fault Models. In *Proc. IEEE International Test Conference (ITC)*, pages 1–8. doi:10.1109/TEST.2011.6139151.

[Hapke et al., 2011b] Hapke, F. et al. (2011b). Gate-Exhaustive and Cell-Aware Pattern Sets for Industrial Designs. In *Proc. IEEE International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, pages 1–4. doi:10.1109/VDAT.2011.5783604.

[Hapke et al., 2012] Hapke, F. et al. (2012). Cell-Aware Production Test Results from a 32-nm Notebook Processor. In *Proc. IEEE International Test Conference (ITC)*, pages 1–9. doi:10.1109/TEST.2012.6401533.

[Hapke et al., 2014a] Hapke, F. et al. (2014a). Cell-Aware Experiences in a High-Quality Automotive Test Suite. In *Proc. IEEE European Test Symposium (ETS)*, pages 1–6. doi:10.1109/ETS.2014.6847814.

[Hapke et al., 2014b] Hapke, F. et al. (2014b). Cell-Aware Test. *IEEE Transactions on Computer-Aided Design*, 33(9):1396–1409. doi:10.1109/TCAD.2014.2323216.

[Hapke and Schlöffel, 2012] Hapke, F. and Schlöffel, J. (2012). Introduction to the Defect-Oriented Cell-Aware Test Methodology for Significant Reduction of DPPM Rates. In *Proc. IEEE European Test Symposium (ETS)*, pages 1–6. doi:10.1109/ETS.2012.6233046.

[Howell et al., 2018] Howell, W. et al. (2018). DPPM Reduction Methods and New Defect Oriented Test Methods Applied to Advanced FinFET Technologies. In *Proc. IEEE International Test Conference (ITC)*, pages 1–10. doi:10.1109/TEST.2018.8624906.

[Hu et al., 2020] Hu, M. et al. (2020). Tightening the Mesh Size of the Cell-Aware ATPG Net for Catching All Detectable Weakest Faults. In *Proc. IEEE European Test Symposium (ETS)*, pages 1–6. doi:10.1109/ETS48528.2020.9131567.

[Hughes, 1988] Hughes, J. (1988). Multiple Fault Detection Using Single Fault Test Sets. *IEEE Transactions on Computer-Aided Design*, 7(1):100–108. doi:10.1109/43.3137.

[Hughes and McCluskey, 1984] Hughes, J. and McCluskey, E. (1984). An Analysis of the Multiple Fault Detection Capabilities of Single Stuck-at Fault Test Sets. In *Proc. IEEE International Test Conference (ITC)*.

[Johnson, 1974] Johnson, D. (1974). Approximation Algorithms for Combinatorial Problems. *Journal of Computer and System Sciences*, 9(3):256–278. doi:10.1016/S0022-0000(74)80044-9.

[Kaczer et al., 2003] Kaczer, B. et al. (2003). Experimental Verification of SRAM Cell Functionality after Hard and Soft Gate Oxide Breakdowns. In *Conference on European Solid-State Device Research*, pages 75–78. doi:10.1109/ESSDERC.2003.1256814.

[Karel et al., 2019] Karel, A. et al. (2019). Analytical Models for the Evaluation of Resistive Short Defect Detectability in Presence of Process Variations: Application to 28nm Bulk and FDSOI Technologies. *Journal of Electronic Testing: Theory and Applications*, 35(1):59–75. doi:10.1007/s10836-019-05776-1.

[Karp, 1972] Karp, R. (1972). Reducibility among combinatorial problems. *Proc. a Symposium on the Complexity of Computer Computations*, page 85–103. doi:10.2307/2271828.

[Khatri et al., 2016] Khatri, D. et al. (2016). Simulation Assisted Uncovering and Understanding of Complex Failures in 28nm Microprocessor Devices. In *IEEE International Symposium on the Physical and Failure Analysis of Integrated Circuits (IPFA)*, pages 64–68. doi:10.1109/IPFA.2016.7564249.

[Lapedus, 2018] Lapedus, M. (2018). Big Trouble At 3nm. *Semiconductor Engineering*. `https://semiengineering.com/big-trouble-at-3nm`.

[Ma et al., 1995] Ma, S., Franco, P., and McCluskey, E. (1995). An Experimental Chip to Evaluate Test Techniques Experiment Results. In *Proc. IEEE International Test Conference (ITC)*, pages 663–672. doi:10.1109/TEST.1995.529895.

[Maxwell et al., 2016] Maxwell, P., Hapke, F., and Tang, H. (2016). Cell-Aware Diagnosis: Defective Inmates Exposed in Their Cells. In *Proc. IEEE European Test Symposium (ETS)*, pages 1–6. doi:10.1109/ETS.2016.7519313.

[McCluskey, 1956] McCluskey, E. (1956). Minimization of Boolean Functions. *The Bell System Technical Journal*, 35(6):1417–1444. doi:10.1002/j.1538-7305.1956.tb03835.x.

[McCluskey and Tseng, 2000] McCluskey, E. and Tseng, C. (2000). Stuck-fault tests vs. actual defects. In *Proc. IEEE International Test Conference (ITC)*, pages 336–342. doi:10.1109/TEST.2000.894222.

[Medeiros et al., 2019] Medeiros, G. et al. (2019). Evaluating the Impact of Temperature on Dynamic Fault Behaviour of FinFET-Based SRAMs with Resistive Defects. *Journal of Electronic Testing: Theory and Applications*. doi:10.1007/s10836-019-05784-1.

[Merritt, 2018] Merritt, R. (2018). Cadence, Imec Disclose 3-nm Effort. *EE Times*. `https://www.eetimes.com/document.asp?doc_id=1333016#`.

[Miyase et al., 2002] Miyase, K. et al. (2002). Don't-Care Identification on Specific Bits of Test Patterns. In *Proc. IEEE International Conference on Computer Design (ICCD)*, pages 194–199. doi:10.1109/ICCD.2002.1106769.

[Mocuta et al., 2018] Mocuta, A. et al. (2018). Enabling CMOS Scaling Towards 3nm and Beyond. In *IEEE Symposium on VLSI Technology*, pages 147–148. doi:10.1109/VLSIT.2018.8510683.

[Mohammadat et al., 2015] Mohammadat, M. et al. (2015). Resistive Open Faults Detectability Analysis and Implications for Testing Low Power Nanometric ICs. *IEEE Transactions on VLSI Systems*, 23(3):580–583. doi:10.1109/TVLSI.2014.2312357.

[OpenCores.org, 1999] OpenCores.org (1999). OpenCores.org. `https://www.opencores.org`.

[Pomeranz and Reddy, 2001] Pomeranz, I. and Reddy, S. (2001). Definitions of the Numbers of Detections of Target Faults and Their Effectiveness in Guiding Test Generation for High Defect Coverage. In *Proc. Design, Automation, and Test in Europe (DATE)*, pages 504–508. doi:10.1109/DATE.2001.915070.

[Prahov and Graupner, 2011] Prahov, R. and Graupner, A. (2011). A Comprehensive Workflow and Methodology for Parasitic Extraction. In *Annual Scientific Conference, Ruse, Bulgaria*. `http://conf.uni-ruse.bg/bg/docs/cp11/3.1/3.1-23.pdf`.

[Qayyum et al., 2002] Qayyum, A., Viennot, L., and Laouiti, A. (2002). Multipoint relaying for flooding broadcast messages in mobile wireless networks. In *Hawaii International Conference on System Sciences*, pages 3866–3875. doi:10.1109/HICSS.2002.994521.

[Rabaey et al., 2003] Rabaey, J., Chandrakasan, A., and Nikolic, B. (2003). *Digital integrated circuits- A design perspective*. Prentice Hall.

[Roth, 1966] Roth, J. (1966). Diagnosis of Automata Failures: A Calculus and a Method. *IBM Journal of Research and Development*, 10(4):278–291.

doi:10.1147/rd.104.0278.

[Ryckaert et al., 2014] Ryckaert, J. et al. (2014). Design Technology Co-Optimization for N10. In *Proc. IEEE Custom Integrated Circuits Conference (CICC)*, pages 1–8. doi:10.1109/CICC.2014.6946037.

[Ryckaert et al., 2015] Ryckaert, J. et al. (2015). DTCO at N7 and Beyond: Patterning and Electrical Compromises and Opportunities. In *Proceedings SPIE*, volume 9427. doi:10.1117/12.2178997.

[Ryckaert et al., 2019] Ryckaert, J. et al. (2019). Enabling Sub-5nm CMOS Technology Scaling Thinner and Taller! In *International Electron Devices Meeting (IEDM)*, pages 29.4.1–29.4.4. doi:10.1109/IEDM19573.2019.8993631.

[Santosh Malagi, 2019] Santosh Malagi (2019). Cadence Cell-Aware Test Rapid Adoption Kit. `http://support.cadence.com`.

[Schaldenbrand et al., 2020] Schaldenbrand, A. et al. (2020). Improving Test Coverage and Eliminating Test Escapes Using Analog Defect Analysis. `https://www.cadence.com/content/dam/cadence-www/global/en_US/documents/tools/custom-ic-analog-rf-design/legato-reliability-wp.pdf`.

[Schaller, 1997] Schaller, R. (1997). Moore's law: past, present and future. *IEEE Spectrum*, 34(6):52–59. doi:10.1109/6.591665.

[Scott Davidson, 1999] Scott Davidson (1999). ITC99 Benchmark Home Page. `https://www.cerc.utexas.edu/itc99-benchmarks/bench.html`.

[Shen et al., 1985] Shen, J., Maly, W., and Ferguson, F. (1985). Inductive Fault Analysis of MOS Integrated Circuits. *IEEE Design & Test*, 2(6):13–26. doi:10.1109/MDT.1985.294793.

[Sherazi et al., 2016] Sherazi, S. et al. (2016). Architectural Strategies in Standard-Cell Design for the 7 nm and Beyond Technology Node. *Journal of Micro/Nanolithography, MEMS, and MOEMS*, 15(1):013507. doi:10.1117/1.JMM.15.1.013507.

[Sherazi et al., 2019] Sherazi, S. et al. (2019). Standard-cell design architecture options below 5nm node:The ultimate scaling of FinFET and Nanosheet. In *Proceedings SPIE Advanced Lithography*. doi:10.1117/12.2514569.

[Shinogi et al., 2001] Shinogi, T. et al. (2001). Faulty Resistance Sectioning Technique for Resistive Bridging Fault ATPG Systems. In *Proc. IEEE Asian Test Symposium (ATS)*, pages 76–81. doi:10.1109/ATS.2001.990263.

[Singh, 2016] Singh, A. (2016). Cell Aware and Stuck-Open Tests. In *Proc. IEEE European Test Symposium (ETS)*, pages 1–6. doi:10.1109/ETS.2016.7519316.

[Sinha et al., 2017] Sinha, A. et al. (2017). DFM-Aware Fault Model and ATPG for Intra-Cell and Inter-Cell Defects. In *Proc. IEEE International Test Conference (ITC)*, pages 1–10. doi:10.1109/TEST.2017.8242059.

[Smith, 1985] Smith, G. (1985). Model for Delay Faults Based upon Paths. In *Proc. IEEE International Test Conference (ITC)*, pages 342–349.

[Stanford EE Handouts, 2001] Stanford EE Handouts (2001). SPICE Quick Reference Sheet v1.0. `https://web.stanford.edu/class/ee133/handouts/general/spice_ref.pdf`.

[Tang et al., 2017] Tang, H. et al. (2017). Using Cell Aware Diagnostic Patterns to Improve Diagnosis Resolution for Cell Internal Defects. In *Proc. IEEE Asian Test Symposium (ATS)*, pages 231–236. doi:10.1109/ATS.2017.51.

[Vaidyanathan et al., 2012] Vaidyanathan, K. et al. (2012). Design and Manufacturability Tradeoffs in Unidirectional and Bidirectional Standard Cell Layouts in 14 nm Node. In *Proceedings SPIE*, volume 8327. doi:10.1117/12.916104.

[Venkataraman et al., 2004] Venkataraman, S. et al. (2004). An Experimental Study of N-Detect Scan ATPG Patterns on a Processor. In *Proc. IEEE VLSI Test Symposium (VTS)*, pages 23–28. doi:10.1109/VTEST.2004.1299221.

[Wadsack, 1978] Wadsack, R. (1978). Fault modeling and logic simulation of CMOS and MOS integrated circuits. *The Bell System Technical Journal*, 57(5):1449–1474. doi:10.1002/j.1538-7305.1978.tb02106.x.

[Waicukauski et al., 1987] Waicukauski, J. et al. (1987). Transition Fault Simulation. *IEEE Design & Test*, 4(2):32–38. doi:10.1109/MDT.1987.295104.

[Weckx et al., 2019] Weckx, P. et al. (2019). Novel Forksheet Device Architecture as Ultimate Logic Scaling Device towards 2nm. In *International Electron Devices Meeting (IEDM)*, pages 36.5.1–36.5.4. doi:10.1109/IEDM19573.2019.8993635.

[Williams and Angell, 1973] Williams, M. and Angell, J. (1973). Enhancing Testability of Large-Scale Integrated Circuits via Test Points and Additional Logic. *IEEE Transactions on Computers*, C-22(1):46–60. doi:10.1109/T-C.1973.223600.

[Williams and Brown, 1981] Williams, T. and Brown, N. (1981). Defect Level as a Function of Fault Coverage. *IEEE Transactions on Computers*, C-30(12):987–988. doi:10.1109/TC.1981.1675742.

[Yang et al., 2014] Yang, F. et al. (2014). Silicon Evaluation of Cell-Aware ATPG Tests and Small Delay Tests. In *Proc. IEEE Asian Test Symposium (ATS)*, pages 101–106. doi:10.1109/ATS.2014.29.

[Zhang et al., 2014] Zhang, F., Thornton, M., and Dworak, J. (2014). When Optimized N-Detect Test Sets are Biased: An Investigation of Cell-Aware-Type Faults and N-Detect Stuck-At ATPG. In *IEEE North-Atlantic Test Workshop (NATW)*, pages 32–39. doi:10.1109/NATW.2014.15.

About the Author

Profile

Zhan Gao was born on January 1, 1991 in Henan, China. She received a B.Sc. in Microelectronics from Xidian University, Xi'an, China in 2013, and an M.Sc. in Microelectronics and Solid State Electronics from the University of Chinese Academy of Sciences in 2016. In September 2016, she became a PhD candidate in the Electronic Systems Group at Eindhoven University of Technology, the Netherlands. Her research interests include logic test, automatic test pattern generation (ATPG), and analog defect simulation. She won the Best Paper Award at the IEEE Latin-American Test Symposium (LATS) 2019, and was nominated for the Best Paper Award at the IEEE European Test Symposium (ETS) twice in 2020 and 2021, respectively.

Presentations in Academic and Industrial Events

1. **Z. Gao**, J. Huisken, K. Goossens, V. Chickermane, J. Swenton, C. Wisnesky, and E.J. Marinissen. "Test for Internal Defects in Library Cells". *The Conference for Dutch ICT-Research (ICT.Open)*. Amersfoort, the Netherlands, March 2017.
2. **Z. Gao** and E.J. Marinissen. "Cell-Aware Test: Significant Test Quality Improvement at Affordable Cost". *imec Partner Technical Week (imec PTW)*. Leuven, Belgium, April 2018.
3. **Z. Gao**, J. Huisken, K. Goossens, C. Wisnesky, J. Swenton, S. Malagi, and E.J.

Marinissen. “Cell-Aware Test: Significant Test Quality Improvement at Affordable Cost". *Cadence User Conference (CDN Live)*, Munich, Germany, May 2018.

4. **Z. Gao** and E.J. Marinissen. “Defect-Location Identification for Cell-Aware Test Using Parasitic Extraction". *imec Partner Technical Week (imec PTW)*. Leuven, Belgium, October 2018.
5. **Z. Gao**, M. Hu, R. Baert, B. Chehab, J. Swenton, S. Malagi, J. Huisken, K. Goossens, and E.J. Marinissen. “Application of Cell-Aware Test on an Advanced 3nm Technology Library". *Cadence User Conference (CDN Live)*, Munich, Germany, May 2019.
6. R. Metcalfe, **Z. Gao**, and E.J. Marinissen. “Cadence’s Cell-Aware Test on imec’s iN5 Technology". *imec Partner Technical Week (imec PTW)*. Leuven, Belgium, October 2019.
7. **Z. Gao** and E.J. Marinissen. “Weak-Fault Coverage Optimization for Cell-Aware Test". *imec Partner Technical Week (imec PTW)*. Leuven, Belgium, April 2020.

Research Activity: Supervising Master Students

1. Santosh Malagi, master thesis: “Library Characterization for Cell-Aware Test", Delft University of Technology, January 2018 – September 2018.
2. Min-Chun Hu, master thesis: “Improving Cell-Aware ATPG Quality on Catching Weakest Faults", National Tsing Hua University, January 2019 – October 2019.
3. Francesco Lorenzelli, master thesis: “Development of Innovative Algorithms and Tools to Test Advanced Technology Cell Libraries", University of Bologna, March 2020 – October 2020.

Marketing Video for Cadence:

1. Z. Gao. “imec, TU Eindhoven, and Cadence – Advances in Defect Location Identification". Munich, Germany, May 2019. `https://www.youtube.com/watch?v=NtwfesYdz90&t=1s`

Blogs for Cadence:

1. A. Klotz. “From China through Cadence to IMEC". April 2017. `https://community.cadence.com/cadence_blogs_8/b/can/posts/from-china-through-cadence-to-imec`
2. P. McLellan. “CDNLive: Testing Times in Munich". May 2018. `https://community.cadence.com/cadence_blogs_8/b/breakfast-bytes/posts/cdnlive-emea-2018-2`
3. A. Klotz. “Best Paper Award at LATS2019 for Zhan Gao". April 2018. `https://community.cadence.com/cadence_blogs_8/b/can/posts/best-paper-award-at-lats2019-for-zhan-gao`

4. P. McLellan. “Cell-Aware Test: Research Cooperation Between Cadence, imec, and TU Eindhoven...Now Shipping in Modus DFT Software Solution". June 2019. https://community.cadence.com/cadence_blogs_8/b/breakfast-bytes/posts/cellawaretest