國立交通大學電機學院電子工程研究所
碩士論文
Department of Electronics Engineering
College of Electronic Engineering
National Chiao Tung University
Master Thesis

瑕疵導向的工業用測試流程
Defect-Oriented Test-Generation Flow for Industrial Cases

盧敬和
Ching-Ho Lu

指導教授：趙家佐博士
Advisor: Chia-Tso Chao, Ph.D.

中華民國 103 年 9 月
Sep, 2014

國立交通大學碩士學位論文
口試委員會審定書

瑕疵導向的工業用測試流程
Defect-Oriented Test-Generation Flow for Industrial Cases

本論文係盧敬和君 (0150241) 在國立交通大學電子工程研究所完成之碩士學位論文，於民國 103 年 9 月 14 日承下列考試委員審查通過及口試及格，特此證明

口試委員：

_____

_____ _____

_____ _____

_____ _____

_____ _____

所　長：_____

摘要

由於現代科技產品對於可靠性的要求大為提升，且傳統的測試方法如 stuck at, transition, 或 small delay 有其測試的盲點，新的測試技術例如 Gate-Exhaustive, N-Detect, 或 Cell aware testing 不斷的被提出。雖然這些新技術可以提升錯誤的涵蓋率，但是也會大量的增加測試的成本，也就是測試的資料數量。在這邊論文中，提出了瑕疵導向的測試方法。本方法會先分析實際電路的設計圖，找出可能的物理損壞，並且透過電路模擬軟體，分析出實際損壞時的類比行為。透過這些分析電路模型，我們將可以透過可靠的商業軟體產生基於瑕疵行為表現的測試資料。最後，結合先前建立的瑕疵表現模型，本流程會分析測試資料，試著刪減測試資料數量。本方法中可以在增加瑕疵測試涵蓋綠的同時，不會增加太多的測試資料。本流程的後半部，亦可直接應用在商用軟體所產生的測試資料上，減少其所產生出來的測試資料數量。本實驗被測試於台灣積體電路的 65 奈米製程，共 845 個標準原件庫上。並且將測試資料刪減的方法，應用在學術，以及工業，電路上，一個電路中將會有超過 180 萬個瑕疵被模擬，並且平均可以刪減 70% 的測試資料。

Abstract

As the modern ICs are facing increasingly tougher DPPM requirements, and the insufficient of the traditional fault model in testing intra-cell defects, new testing techniques such as Gate-Exhaustive, N-Detect, or Cell-aware were proposed. Though the fault coverage can be improved by these methods, the pattern count, which is the main cost of testing, would be significantly increased. In this thesis, a defect aware testing flow for industrial design is presented. The Flow, can target the actual root cause of intra-cell defects while not adding too much patterns and could increase the defect coverage. The rear half of the flow is the proposed methodology which can be applied independently to the state-of-the-art commercial tool to reduce the generated patterns. The newly proposed pattern reduction methodology and the cell-aware testing flow have been evaluated with tsmc 65nm technology on 845 library cells and both ISCAS and real industrial design with up to 1.8 million defects. The experiment result shows an average reduction of 70% in the pattern count without sacrificing the defect coverage.

Acknowledgements

First of all, I would like to express my gratitude to all those who helped me during the writing of this thesis. I' m glad to thank my supervisor Chia-Chao Tso to give me such a challenging and novel topic for my master degree. Then, the most I want to thank are my partners YU-HAO HUANG, and Ting-Chou Lin, who helped me a lot on conducting the experiments and built lots of useful tools and programs. I also appreciate Ying-Yen Chen and Chao-Web Tzeng in RealTek whom helped me on building the industrial testing flow, while they are very busy on work. This two years in NCTU would be so tough without Doctor Hao-Yu Yang, Shih-Hua Kuo who gave me so much instructive advice as well as the useful suggestions. And I would like to thank all the members in the VLSI testing lab for all the great things we have been together. At the end, I want to thank Steve Jobs for building Macbook that is so powerful to do any thing including all the programming works in this thesis and writing the thesis it self.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Due to the high reliability requirements on the modern ICs, such as the in car device, traditional fault models shows their shortage in intra-cell defect detecting, since the models like stuck-at, transition, and small delay, assume the faults only occur between the cells, and this can't model the real behavior of the cells. In the Cell-Aware ATPG flow proposed in [3], the cell-aware method approaches the root cause of the defects, and simulates the analog behavior of the defected cells.

In this Thesis, we first implements the flow proposed in [3]. The flow starts with the RC-extraction of the real cell layout for the parasitic net list as an input of the defect injecting process. The defect injecting process analyzes the parasitic network, and finds the possible locations to add the intra-cell bridges and opens. Then an exhaustive analog simulation is conducted to both fault free and faulty net lists to find the faulty constrains of the cells. At the end, the simulation result will be synthesized into the User Defined Fault Model so the state-of-the-art EDA tool can do the ATPG.

After generating the patterns by the tool, the reduction process will be performed by our proposed pattern reductions methodology. In this methodology, we use the whole defect behavior date of the simulation to find the redundant patterns, which can detect new defects.

Chapter 2

Pervious Work

Due to the traditional fault models assume the defects only occur between the cells, those fault models can't simulate the real fault behavior in a design. The main purpose of this chapter is to introduce the pervious work on detecting the un-modeled fault types. First the insufficient of traditional fault model will be discussed, than followed by the introduction of previous works and why they are not that good to use.

2.1 Problems with traditional fault models

Physical defects such as bridge or open might occurs in any steps during manufacturing of the fabrication process. Conventional way to detect these defects such as stuck-at, transition fault are proven insufficient [3] [6] to detect the intra-cell defects in previous works. To demonstrate the shortage of stuck-at fault, we use a MUX2 as an example. Fig.3.1 is a MUX2 with a table indicating the detected SA faults. The rows refers to the input pattern of this MUX2, and the column stands for the fault name, D1S1 mean pin D1 stuck-at 1. And the cells marked in black means the defect can be detected by the input pattern.

In this example, pattern 001, 010, 100, 101 can cover all the SA faults in this cell. Because the ATPG tool will drop the detected faults, the pattern generator will not generate other patterns cause all faults are detected. What if, some defect in the cell causing the input pattern 111 to fail the test? Since stuck-at fault model the fault between the cells or on the

nets, it is impossible to model this fault constrains.

2.2 Approaches on detecting intra-cell defect

Several works are published to locate the bridge and open defects, In [2], over 1 million tested device showed the defects are intra-cell defects and are only testable by embedded multi-detect (EMD) [4] patterns but not covered by stuck-at fault patterns. Method used in industry such as N detect[10] are proven[5] to hit the intra-cell defect by chance. And the application of gate-exhaustive testing to detect the intra-cell defects is performed in [1]. However, these methodologies are either too complex or cost too much patterns in real designs. Work [13] shows a better way to fill the unknowns to increase the test coverage over the intra-cell defects, but only slightly improved the probability. [3] presents a new methodology to directly target the layout-base intra-cell defects, and generates the pattern that increase significantly the defect coverage by state-of-the-art atpg tool. In contrast to other techniques, the defect modeling methodology targets the actual root cause of the cell layouts. And this is the methodology we will use to test the intra-cell defects and try to reduce the pattern counts. Some experiment results on the industrial processor of this methodology are conducted in [7].

Chapter 3

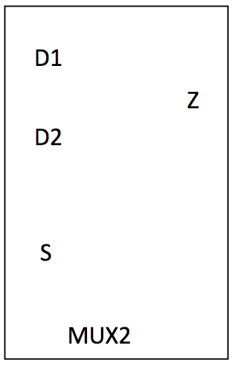
Introduction to Cell-Aware testing

Due to the requirement of high reliability of the ICs, new testing methodology had been proposed. One of them is cell-aware testing. In this section, we will briefly introduce the cell-aware testing and left the implementation details in the coming sections.

3.1 What is cell aware testing

Cell-aware testing is a testing method that generates the patterns base on the real defects in the layout of a standard cell. Due to the lack of the ability of testing the intra-cell defects, some intra-cell testing methods are proposed, such as Butterfly structure[9], Effect-cause[11], and Defect Aware Test Pattern[12]. But all these method use indirect method or detect the defect. In the Defect-Oriented Cell-Aware ATPG[3], the author analyze the real layout and simulate the syndrome of the defect analogly to rebuild the true fault behavior. As we believe this is a good solution to detect the intra-cell defects. The theorem will be use in this thesis, but modified some implement details to fit out research. The cell-aware testing will all refer to the theorem proposed in [3] in the later thesis.

In the rest of this chapter, we will demostrate why cell-aware is better than inter-cell fault models, then briefly describe the cell-aware ATPG flow, and will end at the problem we found in this ATPG flow.



D1	D2	S	Z	D1S1	D1S0	D2S1	D2S0	SS0	SS1	ZS0	ZS1
0	0	0	0	■							■
0	0	1	0			■					■
0	1	0	0	■					■		■
0	1	1	1				■	■		■	
1	0	0	1		■				■	■	
1	0	1	0				■	■			■
1	1	0	1		■					■	
1	1	1	1				■			■	

Figure 3.1: MUX2 and the Stuck-at fault detect map, D1S1 stands for D1 stuck-at 1

3.2 Why Cell aware is good

State-of-the-art fault models such as Stuck-at(SA), Transition-Faults, and small-delay defects are base on the assumption that the fault only occurs between the cells, the port of a cell, or on the nets connecting the cells. Lets take a look at a MUX2 as an example of how insufficient will SA fault be.

3.2.1 Cell aware example on MUX2

Fig.3.1 is a MUX2 with a table indicating the detected SA faults. The rows refers to the input pattern of this MUX2, and the column stands for the fault name, D1S1 mean pin D1 stuck-at 1. And the cells marked in black means the defect can be detected by the input pattern.

In this example, pattern 001, 010, 100, 101 can cover all the SA faults in this cell. Because the ATPG tool will drop the detected faults, the pattern generator will not generate other patterns cause all faults are detected. What if, some defect in the cell causing the input pattern 111 to fail the test? Because cell-aware testing is base on the real possible defect simulating, no fault cause by the possible defect will be left behind, and in this case, 111 will sure be generated.

3.2.2 Flexibility of defect models

The best part of this testing structure is the flexibility to add the new defect types, or modify the insertion method of the existing one. For example, we can add the net-open as

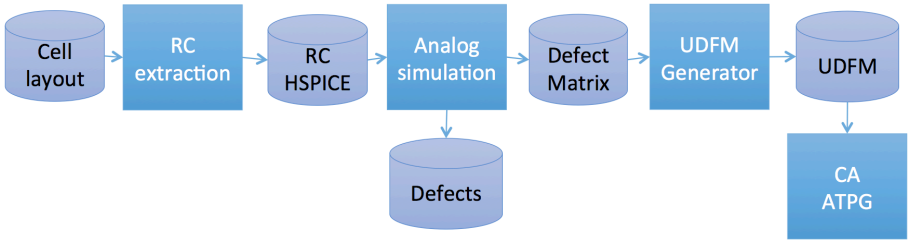


Figure 3.2: A simple flow of cell aware ATPG

well as bridging defects in or between the nets of each cell to simulate the defects in the FAB. Or even modify the parameters of the transistor to simulate the process variation. One can add a new discovered defect type to the flow to improve the DPPM without any effort.

3.3 Cell aware testing flow

The complete flow of the cell-aware ATPG will be introduced in this section. As the Fig.3.2 shows, the Flow starts form the RC extraction of the real layout of the standard cell, and end with generating the UDFM. This section is only to give the reader a quick pick up with the concept, the detail implementation will be discuss later.

3.3.1 Concept of UDFM

User Define Fault Model, UDFM in abbreviation, is an interface provided by the commercial EDA tools, allowing the user to add fault models with the testing constrains. By using UDFM in the ATPG flow, one can not only generate patterns to test the common fault models, but also the complex faults hard to generated by traditional fault models. The detail format and the generating skills will be introduced in the later paragraph; we now focus on the basic concept of the UDFM so reader can understand the coming sections.

UDFM contains three main parts, fault type, fault behavior, and activating conditions. First, UDFM support two kinds of fault type, static fault and delay fault. Static fault is something like stuck-at fault, the delay fault have the concept of transitioning, just like

transition or small delay fault. The fault behavior describe the faulty value at the output pin, and the activating constrain is the input combinations which activate the defect and result in a fault. An example of MUX2 is provided below:

Cell “MUX2” Fault “Z1”

test StaticFault “Z” =1;Condition “D0” =0, “D1” =0, “S” =0;

test StaticFault “Z” =1;Condition “D0” =0, “D1” =1, “S” =0;

test StaticFault “Z” =1;Condition “D0” =0, “D1” =0, “S” =1;

In this example, we can know that static fault Z stuck-at 1,while the input pattern equals to 000, 010, or 001. After generating the UDFM, the ATPG tool will use the information in the UDFM to generate the patterns at the primary input. Just as the way it treated the traditional fault model.

3.3.2 Layout extraction

To find the possible defect in the cell, real layout of the cell is necessary. By extracting the layout of the cell, the detailed parasitic information will be obtained, the informatoin as a reference of how to add defects and simulate the behavior of a defected cell.

3.3.3 Adding defects

This is the most important part of the flow, since the size, location, or the type of the defect should be as close as to the real condition. In this work, only intra-cell bridge was added to the cells, but any type of defect that might occur can be added to improve the DPPM.

First of all, is where to add the bridge? A more sophisticated method is to analyze the layout file such as DEF, or GDSII, and add a resister between new net that close enough to each other. But it takes a large effort to do so, since every bridge added to the cell, the extraction process must be done again to get the new parasitic net list. A more efficient way to locate a bridge site, is to look at the parasitic net list. Since the coupling capacitor only occurs while two layouts are close enough, or the size if large enough, we can use

this as a method to find the possible locations of the bridge fault.

In our work, we calculated the possible bridge size of tsmc 65 nm process technology by adding a smallest metal that doesn't break DRC rules between two nets and extract the layout to get the value of a bridge resister. As a result, we add a resister of 1 ohm as a bridge in this thesis.

3.3.4 Analog simulation

After generating the defected net list, the analog simulation is performed. The analog simulaton can give us an accurate image of the true behavior of a faulty cell, and generate the input constrain to activate the fault. Since our goal of this thesis is to generate the static fault. We use the DC static simulation to analyze the fault. In the future work, transition, and small delay faults would be added to detect the delay faults.

3.3.5 Defect Table

Before building UDFM, we need to know what input constrain will result in failure. There are plenty of ways to find the constrains, in the cell-aware testing, constrains are generated by the simulation results. In this thesis, we record the simulation result in defect tables, which the rows stand for the input constrain, the columns refer to the defects, and the cells marked if it is detected. A simple example is shown in Fig.3.3. In the first row, a ”D” was denoted, and it main the cell will fail if defect ”d4” exist and pattern 000 was injected.

3.3.6 Building UDFM

With the defect table above, we can know what constrain might result in failure, so the basic version can be built. But generating the whole table will result in gate-exhaustive test, we to reduce the table first. Consider a condition in Fig.3.3, pattern 010 includes all the defects 000 have, that is, we need not to test 000 if 010 was tested. By doing column coupling, marking out the defects been included, we are able to reduce the table into a

STIMULI	d1	d2	d3	d4	d5	d6	d7	d8
000				D				D
001		D			D	D	D	D
010	D			D			D	D
011						D		
100			D	D				D
101		D				D		
110	D		D	D			D	D
111	D		D				D	

Figure 3.3: A simple example of defect table

smaller one containing less conditions, but remain the ability to test all the defects, as shown in Fig.3.4. In this example, by generating the UDFM with 001 and 110, we are able to detect all defects in this cell.

3.3.7 Generating patterns with UDFM

After generating the UDFM for all the cells in standard library, the final step is to generate the patterns to test the design. First, the tool will assign each cells in the design a UDFM, then try to back trace the design and find the patterns at the primary input to meet the UDFM constrain. After generating a pattern, the tool will simulate the pattern, and mark out the detected faults, so the same constrain won't be generated twice.

By using the UDFM, we are able to add the self-defined fault models. But we found a problem that would result in redundant patterns if using the UDFM.

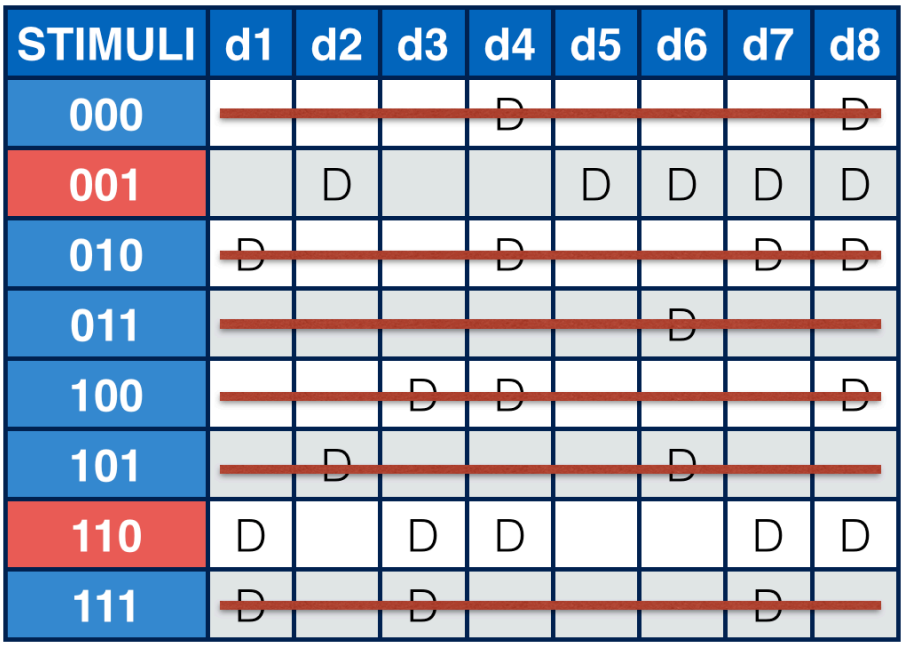


Figure 3.4: Reduce the defect table by column coupling

Chapter 4

Pattern Reduction

This chapter is the main theory of this thesis. By generating the patterns with UDFM build from the defect table, we see the possibility of generating redundant patterns. The coming sections will describe the reason of producing the redundant patterns and how we reduce the pattern without sacrificing any defect coverage.

4.1 Problems with UDFM

The defect table reduction process is necessary in building the UDFM. Without the reduction process, the UDFM will includes all the input constrains and results in gate-exhaustive test. But it processes it the main reason that using UDFM will result in redundant patterns. First, UDFM doesn't contain any defect information, and so the atpg tool won't know which defect was actually detected while generating patterns. Furthermore, the fault simulating process after every pattern for fault dropping won't know which defect is dropped. As an example in Fig.5.7, assume the constrains mark with red in the defect table is the two constrains of UDFM100, and the patterns on the left are all generated from different UDFM. If we keep the defect table while fault simulation, we can see that Pattern 98 and Pattern 99 accidentally activate some defects in the defect table of UDFM 100. And the defects activated by Pattern 98 and Pattern 99, which is, activating the constrain 010 and 100, can cover all the defects in the constrain 110. Because Pattern 100 is generated from the UDFM constrain 110, and it has already been tested by the previous patterns, we can

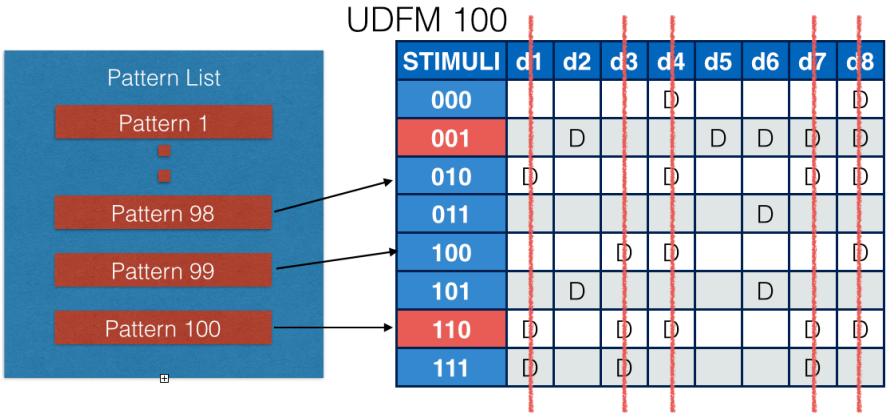


Figure 4.1: The possible redundant patterns

claim this pattern redundant. All these defect dropping works needs the whole information of a defect table, but the previous cell-aware flow didn't include the information in the ATPG flow, and this is what makes we different.

4.2 Pattern reduction flow

The reason of generating the redundant pattern is the lack of defect information while doing ATPG. Even if we build the UDFM into gate-exhaustive, which means all cell input combinations are considered necessary while ATPG, the problem still exists (since no one knows what defects had been dropped). So we now proposed a new pattern reduction method base on the defect aware fault simulations. Defect Aware Fault Simulation simulatie the patterns, and then drop all the detected and propagated defects. By this technique, we are now able to know which exactly the defects are detected by the pattern.

As the flow shown in Fig.4.2, after generating the patterns with state-of-the-art commercial tool, we simulate the patterns with the defect aware fault simulator me made by our self. The defect aware fault simulator will drop the detected defects in each pattern. If any pattern can't detect any new defects, we declare this redundant, and delete it from the pattern list. Next section will introduce our defect aware fault simulation flow.

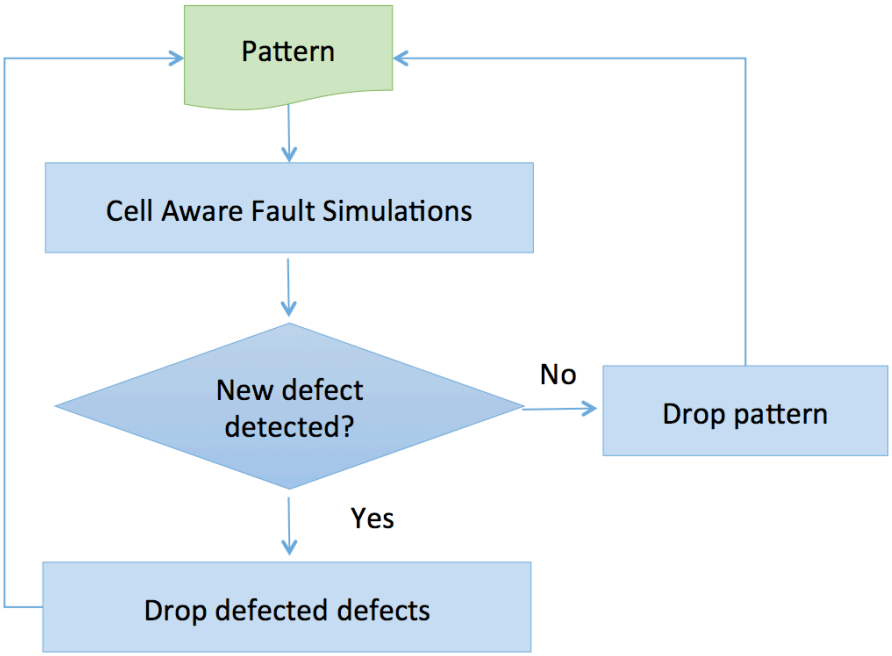


Figure 4.2: The proposed method to reduce the test patterns

4.3 Defect aware fault simulation

Defect aware fault simulator, the simulator in abbreviation, is a self-made cell-aware fault simulator which can do the fault simulation and drop the detected defects. The first step of the simulator is to build a defect table for all cells in the design. Then, a pattern is simulated to find which cell-aware fault can be detected. Third, drop the defects in the defect table with the input patterns of each detected cells as the constrain of the defect tables. As an example, Fig.4.3 simulates an input pattern 1101 at the primary input. And find the faults at AND1 and OR can be detected (AND2 can not propagate). Defects in the defect tables above the cells are crossed off by the input patterns of each cells, which is, 11 at AND1 and 10 at the OR gate.

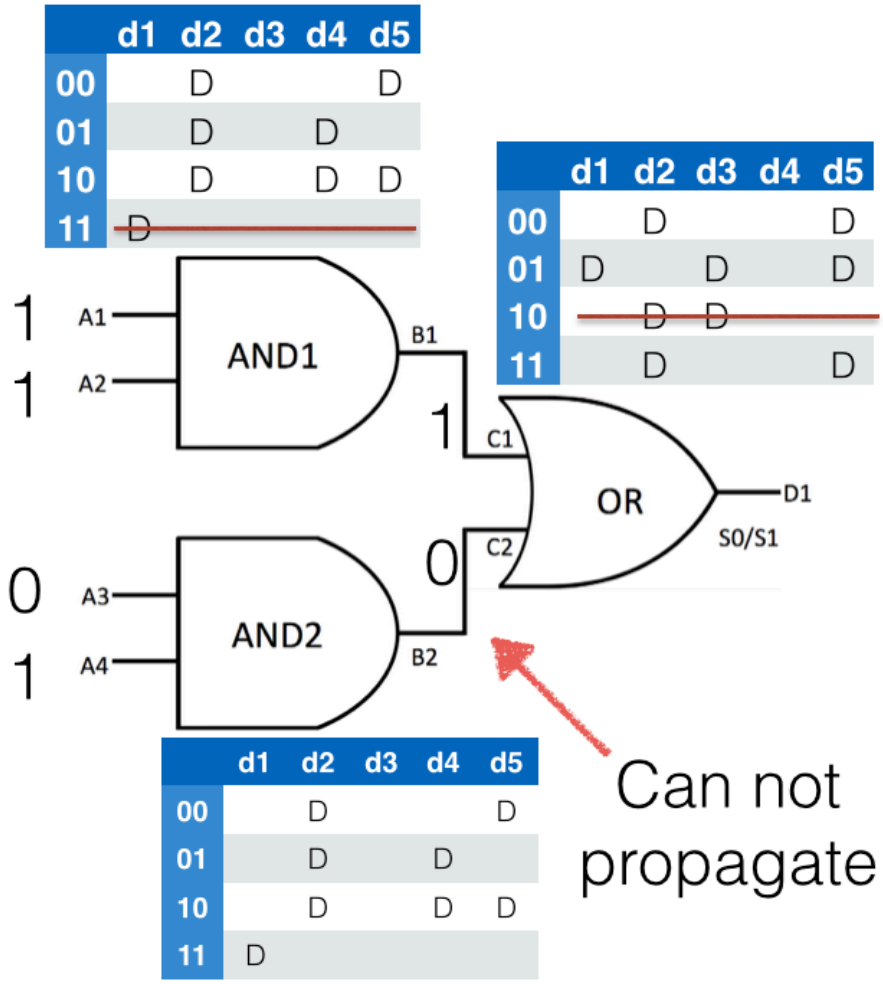


Figure 4.3: The proposed method to reduce the test patterns

Chapter 5

Implementing details

In this section, we will introduce the detailed defect-oriented testing flow, which includes Cell-Aware atpg and the proposed pattern reduction methodology. Firstly, we will introduce the full image of defect aware testing flow, and followed by the detailed description of the remaining steps. And the proposed reduction methodology will be discussed at the end of this chapter.

5.1 Defect-Aware Testing flow

The flow starts with the Cell-Aware ATPG flow, followed by the defect aware fault simulation, and a pattern reduction methodology was shown as an ending. The Fig.5.1 is the full image of the testing flow, a mixture of state-of-the-art tool and new self-made tools. The steps colored in red are the Cell-Aware ATPG flow; the others are the Defect-Aware fault simulation.

5.2 Cell-Aware ATPG

The objective of this section is to introduce the Cell-Aware ATPG based on previous work [3], alone with some implementing details.

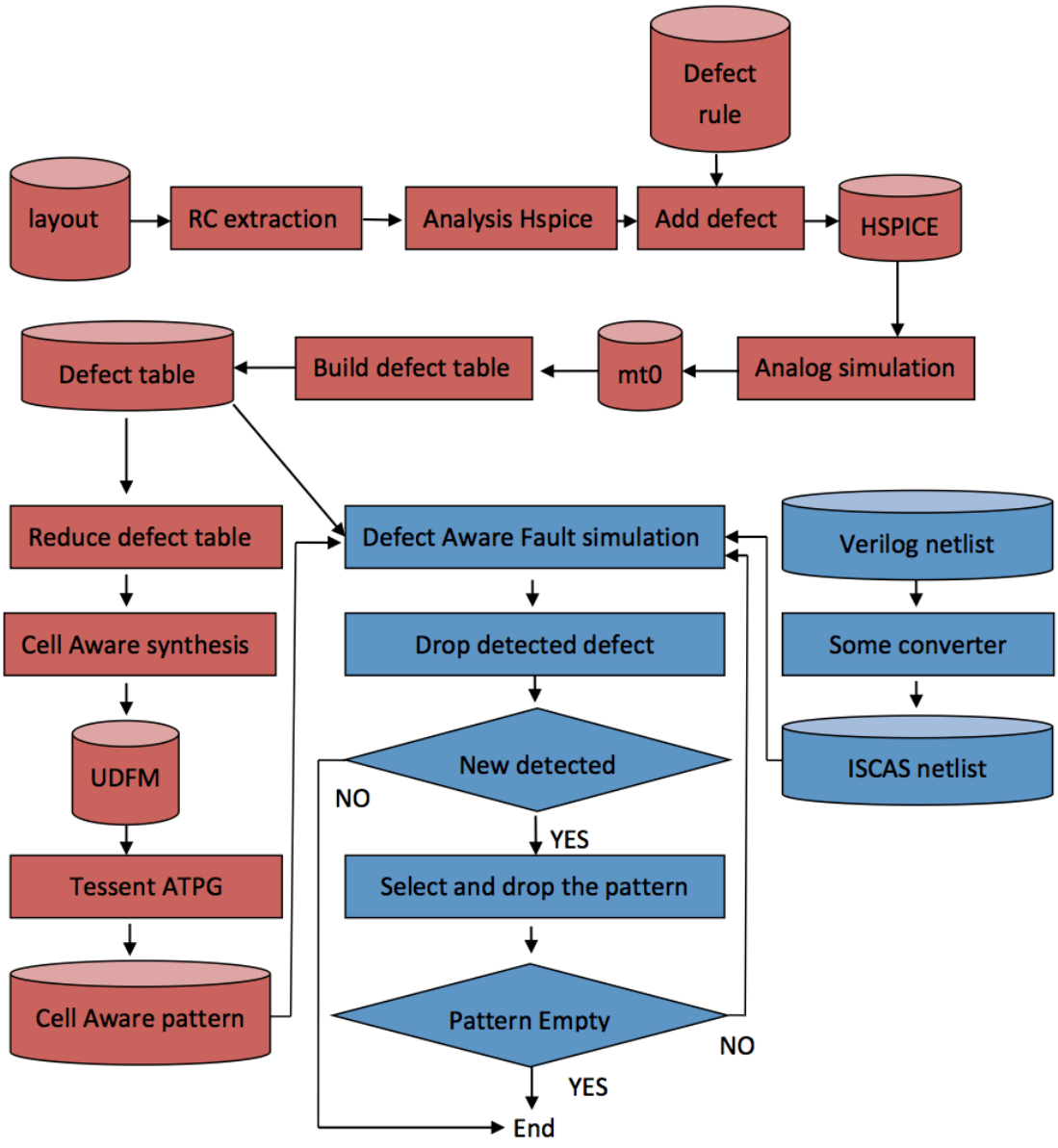


Figure 5.1: Defect Aware Testing Flow

5.2.1 Pervious work

The cell-aware flow in this thesis is based on the methodology proposed by Mentor Graphic, and it starts form the RC extraction of actual layout view of the standard cell. After extracting the layout with state-of-the-art commercial EDA tool, we will obtain the netlist containing RC information, which is use as input for the next step where the defects are injected. Next, the defects are analyzed and injected base on the defect rules, which will be discuss later. Then, the exhaustive analog simulation is performed on each defected netlist, and the results will be stored as the format of defect tables, later be synthesized into the user define fault model, aka UDFM, with the reduced defect table.

5.2.2 Locating defect sites

Two key points in this step, defect locating, and determine the defect size. In this thesis, only intra-cell bridges are considered. First, we locate the defect site with the RC information form the previous step. The parasitic capacitors only occor when two nets in the layout are too close to each other, and this might results in intra-cell bridgings. So we use the capacitor in the parasitic net list to locate the defect sites. When we get a new parasitic network, we analysis all these capacitor, identify the capacitors not in the nets, and are in the same layer. Then connect the nets causing the capacitor with a resistor to simulate the bridge.

5.2.3 Bridge size

After locating the defect sites, a resistor was added. In this thesis, we use the tscm 65nm low power process technology. The size of the resistor is calculated from the technology node of the design. First, we put a metal between the closest nets. The length between and the width of the bridge metal, is decided by the smallest value in the DRC rule. The formula to calculate the resistor is R = C(L/W). The length between the nets is already the smallest, and the width is the largest (or it will be a new net between the nets), the R we calculated by this way would be the smallest value we might have in a real design. Since

the value of the resistance determine the degree of the failure, and the smaller the worse, we believe the bridge added by this way can represent the most severe bridging faults.

5.2.4 Building Defect Table

To build the defect table of each cell, fault free netlist will be simulated for all input combinations as the golden value. All possible defects, including bridging or opening, are simulated exhaustively to find out the effect of each defect. The output would be considered faulty if the voltage diverse from golden value by more than 50% of the supply voltage, and it mean the defect can be detected by the input constrain.
Defect table is a matrix containing the information of detected defects for each input patterns. The rows of the defect table refer to the input combination and the clumns are the possible defect in the layout of each cell. For a clearer visual, please refer to Fig.3.3. In this defect table, the d1 to d8 are the defects of this cell, the 000 to 111 are the exhaustive input combination, and the ”D”s inside the table refer to the detected defects. For instance, the ”D” in the first row stands for the detection of d4 by input 000.

5.2.5 Reduce Defect Table

After building the defect table, the reduction of the defect table will be necessary, or it might results in a gate-exhaustive fault model, containing all input combinations. The reduction concept is very easy, since it only need the simple column coupling. But the implementing is a NP hard problem [14], and heuristic algorithm [8][15] is used in this work. We use Fig.3.4 as an example of the defect table reduction. In this table, we can observe that 000 can be merged into 010 obviously, and so as 011 and 101. By iterating this column coupling process, the remaining table will only contain the minimum input combinations, which cover all the defect detection, and can be easily transformed into UDFM, since only syntax issues need to be handled.

This defect table reduction process is the root cause of the redundant patterns we dis-

cussed previously.

5.2.6 UDFM

UDFM, stands for User Define Fault Model, is the interface to add fault model into state-of-the-art commercial EDA tools to generate patterns based on it. Additional to traditional fault models, users can add the input constrain that activate the fault. As an example shown in Fig.5.2, is an UDFM of an inverter. In the example, a static fault ”ZN”: 1 caused by condition ”I”:0; is performed. One more example is provided as flowing:

Cell “MUX2”

Fault “F1”

test StaticFault “Z” =1;Condition “D0” =0, “D1” =0, “S” =0;

Fault “F2”

test StaticFault “Z” =1;Condition “D0” =0, “D1” =1, “S” =0;

test StaticFault “Z” =1;Condition “D0” =0, “D1” =0, “S” =1;

In this example, F2 will be claim detected if one of the two testing constrains is detected, that is, 010 and 001 can detect this fault. Not only the static fault can be use in the UDFM, delay fault with transforming are also supported. The flowing is an example of a delay fault UDFM:

Cell “MUX2”

Fault “F1”

test

DelayFault ”Z” : 1;

Conditions ”D0” : 10;”D1” : 00; ”S” : 00;

```
UDFM {
   Version:3;
   UdfmType("intra_cell_defects"){
   module ("INV"){
      Fault("ZN"){
         test{
            staticFault{"ZN":1;"}
            Conditions{"I":0;}
         }
      }
   }
   }
}
```

Figure 5.2: Example of User Define Fault Model

This example shows the MUX2 need to be test under the transition mode. The activating constrain is D0 transitioning from 1 to 0 and so on. Than the output will have a small delay on 1.

5.3 Defect-Aware Fault simulation

After the generating cell-aware patterns, the key point of this thesis is to reduce the pattern count by doing defect-aware fault simulations. This section is to present the defect-aware fault simulation in detail. Starting from the description of the program, then the synthesizer we use to handle verilog, and how we inject cell-aware fault to the circuit, and end with the most important defect dropping process.

5.3.1 Brief Description

Defect-Aware Fault Simulation, or, the simulation for short in this section, is a complex mixture flow of state-of-the-art EDA tools and lots of self made tools, parser as well as circuit simulator. The image Fig.5.3 is the brief flow before doing the simulation.

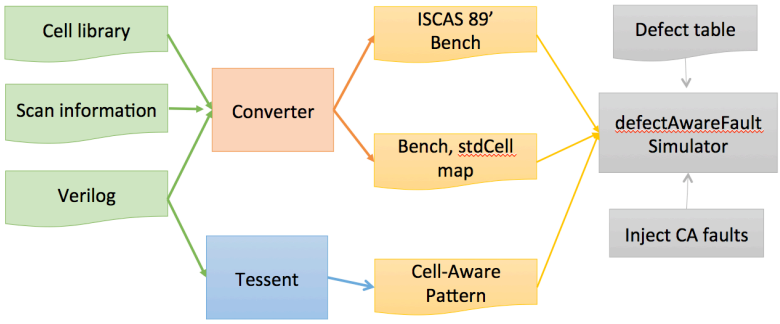


Figure 5.3: Synthesizing file for the simulator

For a better view of the flow, we divide the flow into three parts. The green files are for the converter to synthesis the ISCAS bench, which will later be use as the input circuit for the simulator. The Blue square under the graph is the tool we use to generate the Cell-Aware test patterns, what we want to reduce. The gray box on the right is the main work of this thesis, which can simulate the defect and try to reduce the redundant patterns.

The verilog colored in green, the netlist to be test, is a compiled gate-level netlist with scan chain. The scan information is in the stil format from compiling the verilog, the converter will need this to identify the scan cell, and modify the netlist. Tessent, colored in blue need the compiled verilog and the UDFM generated in the pervious section to generate the Cell-Aware test patterns. The last step, defect aware fault simulator, at the right, reads in the ISCAS 89' bench as the circuit format, and use the bench, standard cell mapping file to identify the I/O pin of the standard cells in the bench circuit.

5.3.2 Bench Synthesizer

Bench Synthesizer, Fig.5.4, is a self made tool to convert the gate-level verilog into the ISCAS format and still preserve the original standard cell information. The tool will first unroll the gate-level verilog so that only top module will be preserved. Due to the design for testing issues, scan chain must be handled. As a result, we break the scan cells apart, let the D connects to the circuit output and Q connect to circuit input directly, as shown in FIG.5.5. By breaking the scan chain and add the input output to control the scan cells, we are now able the control the sequential circuit by inserting the scan in data directly to the

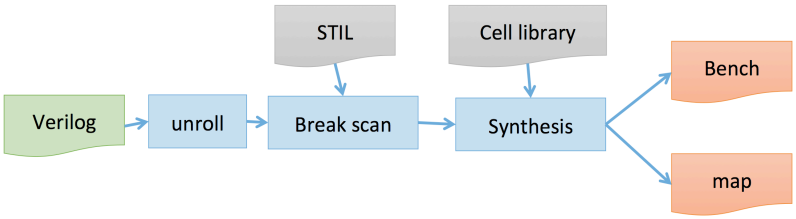


Figure 5.4: Bench Synthesizer Flow

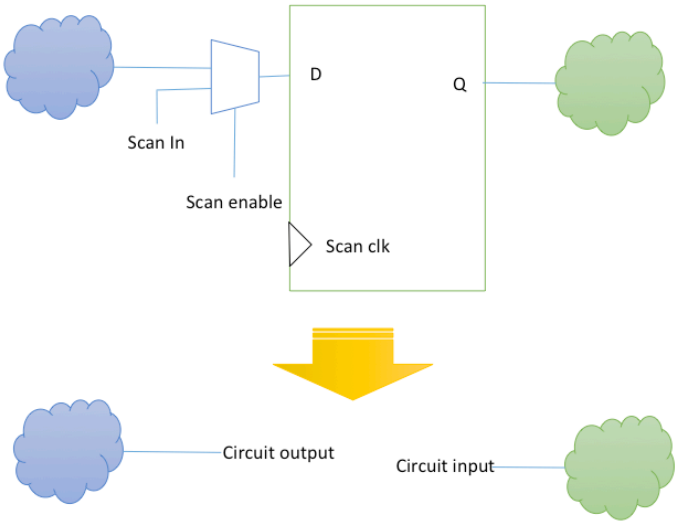


Figure 5.5: Handle the scan chain by connecting scan cells directly to the I/O of the circuit.

circuit in the next stage, and read the scan out data from the output pin without shifting the scan chain. The core engine of the simulator is a simulator only accept primitive gates in ISCAS bench style, so we synthesis the standard cells into primitive gates by using the logic information in the cell library. The final step is to output the circuit in bench format, and a mapping file containing the information of how to map the primitive gate back to the standard cells. With these files, we are now able to simulate the gate level verilog with scan chains.

5.3.3 Injecting Cell Aware Faults

Because we only use static fault in this work, and all fault behavior of static fault can be modeled as stuck at 0 and stuck at 1 at the output of each cells, we simply add the stuck-at 0 and stuck-at 1 at the output of each standard cells, as shown in Fig.5.6. After injecting the faults, we can claim the fault be activated by comparing the input combinations with

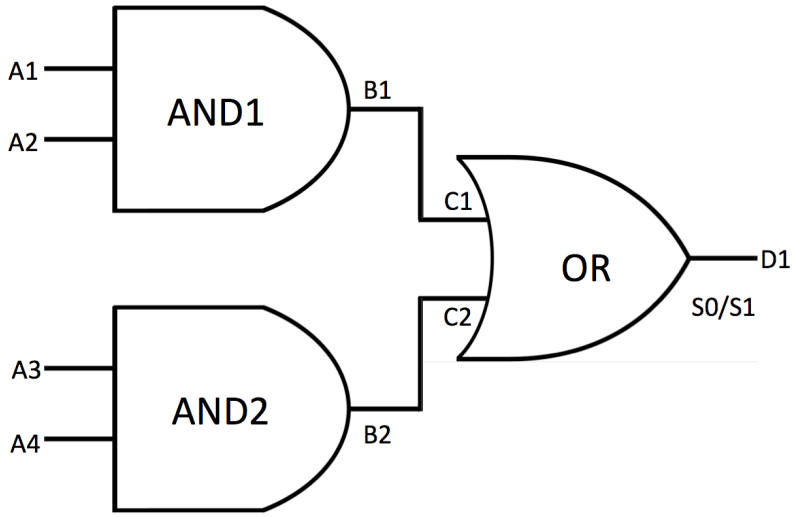


Figure 5.6: And AO21 cell demonstrates the fault injection at the output.

the constrains in the UDFM. For instance, to claim the fault been activated in Fig.5.6, we need to trace back the bench circuit and get the input combinations A1 A2 A3 A4, which is composed by the input of AND1 and AND2. If A1 A2 A3 A4 equals to 0000, and it is defined to be faulty combination, then we can say that the stuck at 1 at the output D1 is activated.

5.3.4 Defect dropping

The main feature of this flow is to reduce the cell-aware patterns generated by UDFM, and is done by finding the pattern combinations that will cover all the defects need to be detected by other patterns. As the Fig.5.7, the pattern 98 and pattern 99 are patterns generated from other UDFM constrains. But in the simulation process, these two patterns will activate the defects in UDFM 100, which use to generate the Pattern 100. The two constrain activated by pattern 98 and 99, which is 010 and 100, can cover all the defects originally designed to be tested by pattern100. As a result, pattern 100 will test the already tested defects. So we propose the flow to reduce the patterns by defect dropping in the defect aware fault simulation process.

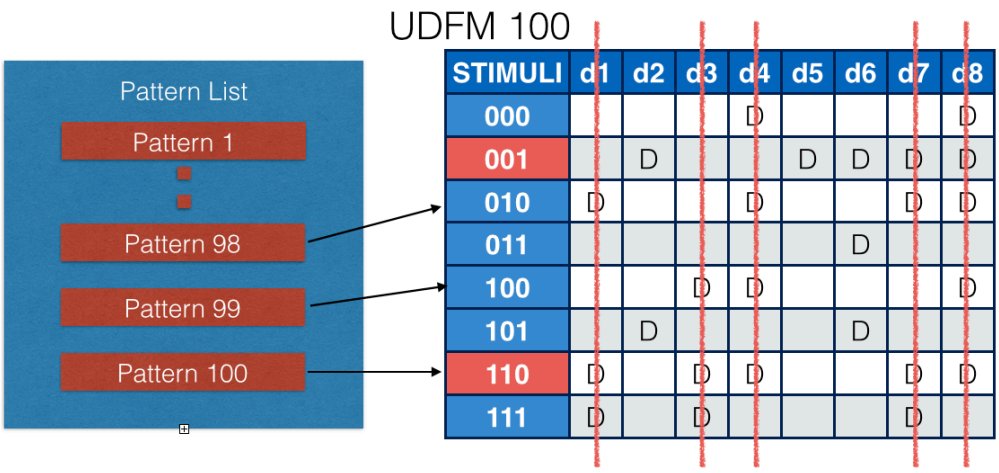


Figure 5.7: UDFM may result in redundant Patterns

Chapter 6

Experiment Results

Two main flows need to be experimented in this thesis, the cell-aware atpg flow, and the pattern reduction method. In the cell-aware testing flow, we want to know if the patterns generated from the traditional fault model really are insufficient to test the cell-aware fault list. Then we tested our pattern reduction method on the cell-aware patterns generated from the state-of-the-art commercial atpg tool, to know if the methodology really works, and can reduce the patterns in what degree. In this chapter, the experiment setting will be the opening then follows by the three experiments conducted.

6.1 Experiment settings

All the experiments or conducted on the work stations provided by RealTek, and all of them are Dual-Core CPU with 3.0 GHZ and 16 GB Memory. Cell-aware atpg flow use the tsmc 65 nm lowpower and tsmc 65nm lowpower high vt as the cell library. We use Tessent by Mentor Graphic as the atpg engine and calibre as the RC extraction flow.

6.2 Experiments

Three experiments are conducted to test the ability of this work. First, we design an experiment to test if the traditional fault model can cover the cell-aware fault lists. Then we modify the cell-aware testing flow to drop some pattern counts but maintain the cover-

	CA Patterns	CA coverage	SA Patterns	SA Pattern Coverage	CA Fault Numbers
S38584	277	91.07%	175	64%	26370
S38417	502	89.06%	152	56.3%	30035
b17	874	94.79%	543	60.93%	19994
b20	351	93.65%	171	66.15%	19994
RTK	4377	80.70%	1552	60.66%	648068

Table 6.1: Stuck-at pattern on Cell-aware Fault List

age. Final experiment is to prove the reduction methodology and try to reduce the pattern counts.

6.3 Stuck-at pattern on cell-aware faults

In this experiment, we will first generate the patterns from stuck-at fault. Then conduct a fault simulation on the cell-aware fault list with the stuck-at fault patterns. And we will find out if the cell-aware fault list does contain the fault that can't be tested by stuck-at faults.

SA Pattern coverage in Table.6.1 is the fault simulation results of stuck-at patterns on cell-aware fault list, and wee can see that only 60% in average can the stuck-at patterns test the cell-aware fault. This experiment shows the insufficient of the traditional fault model for the low DPPM requirement nowadays.

6.4 Drop the cell-aware fault list with stuck-at patterns

In Table.6.1 we can see the SA patterns is much more than CA Patterns in about 2.36 times. This is the reason we want to find some ways to reduce the pattern count. Before our pattern reduction method, we want to drop some patterns with stuck-at patterns, since it is the must-test patterns in the industrial testing flow. So we modify the test flow as shown if Fig.6.1. This atpg flow will stuck-at patterns at first, than drop the cell-aware faults by do fault simulations with stuck-at patterns. After dropping the cell-aware faults, we do the cell-aware atpg to generate the new cell-aware patterns.

By modifying the atpg flow, we can see the great reduction in the cell-aware pattern count,

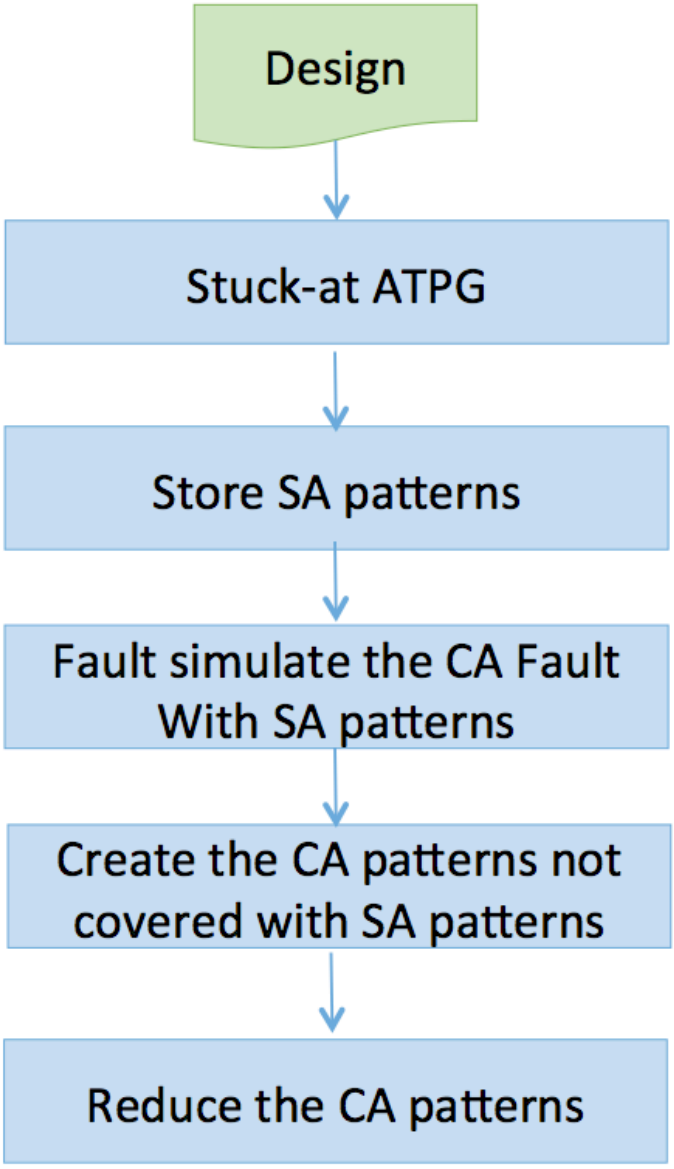


Figure 6.1: Reduce the pattern count with the stuck-at patterns dropping the cell-aware fault list

as shown in Table.6.2. Two things important in this table, first, the new CA patterns have a reductions of 27% in average, second, the fault coverage remains the same or even higher. The reason why the fault coverage will get higher is because stuck-at fault accidentally hit some cell-aware faults that are too hard to test.

	CA Pattern	Original Coverage	New CA Patterns	Final Coverage	Stuck-at Patterns	Total Pattern Reduction
S38584	277	91.07%	173	91.29%	175	23%
S38417	502	89.06%	350	90.52%	152	25%
b17	874	93.04%	352	94.79%	543	36%
b20	351	93.65%	196	94.05%	171	29%
RTK	4377	80.70%	3014	81.07%	1552	22%

Table 6.2: New Cell-aware Pattern count after dropping the fault list with stuck-at faults.

	CA Patterns	Reduced CA Patterns	Proposed Reduction	Reduction Ratio	Run Time
S38584	277	173	128	26.01%	5.88 sec
S35932	118	74	26	64.86%	2.62 sec
S38417	502	350	259	26.00%	4.64 sec
b17	874	352	160	54.54%	12.53 sec
b20	351	196	70	64.28%	4.59 sec

Table 6.3: Proposed reduction method

6.5 Proposed reduction method

This is the experiment of our proposed reduction method. In the experiments, we try to reduce the new cell-aware patterns generated from the pervious experiment. Table.6.3 is the experiment result, first the run time are all in seconds though we need to simulate millions of defects to reduce the patterns. Second, the column ”Proposed Reduction” is the reduced pattern number we achieved. An average of 43.45% can we reduce. The reason why the circuit of RealTek is removed is because our self-made verilog simulator haven’t support decompressor yet.

Chapter 7

Future Work

In the next step of this work, we will focus on supporting other defect types, such as open, transistor defects. By adding other defects the work will be able to enhance the defect coverage and improve the reliability. Next, the analog simulation flow will be modified. Simulations of the defect syndrome in the work are based on stuck-at fault. By adding some timing issues in the simulations, we will be able to generate new ATPG constrains, such as small delay defect, to handle new fault models. Final step would be the research on pattern reordering method. There are two reason to do this improvement. First reason is that we didn't do any pattern reordering in this work, since we haven't support other fault model, and defects, the reordering method might not have the capability of dealing with the new patterns. The other reason is because the current defect table of each pattern, which is comprised of all the defect table of every cell in the design, is too large to handle.

Chapter 8

Conclusion

In this thesis, we complete two main projects. The cell-aware atpg flow, and the pattern reduction method. Stuck-at fault patterns can only achieve 65% of the coverage on out cell-aware fault list produced in this thesis. This stands for the fact that traditional fault model is not sufficient for high reliability requirements in nowadays. The proposed pattern reduction methodology proves the possibility of generating redundant patterns in state-of-the-art atpg tool, and more over achieve a reduction ratio of 43% in the new cell-aware pattern counts, or 70% in the original patterns.

The tools and flows made for this thesis includes, the RC-extraction flow to get the parasitic net list of the cells. A defected HSPICE synthesizer, which automatically reads the parasitic net list, then find and inject the defects into the net list, and synthesize into HSPICE format for analog simulations automatically. Defect table reduction algorithm to reduce the defect table with a heuristic method. Then the UDFM synthesizer reads the reduced defect table and transform into UDFM format. And the most important is the atpg flow that supports cell-aware testing.

In the pattern reduction methodology, the key work is our self-made verilog simulator. The simulator support the ISCAS circuit format, and so we build a ISCAS bench synthesizer transforming the verilog files into the ISCAS format with the logic information in the cell library. To fit the design for testing issues, the scan chain is supported to increase the testability and fasten the simulation time. And all these flow including the cell-aware test and defect aware fault simulation flows are all capable for industrial designs, and

tested with ReakTek communication network IC with tsmc 65nm low power and 65nm low power high vt process technologies.

Bibliography

[1] K. Cho, S. Mitra, and E. McCluskey. Gate exhaustive testing. *Proc. of IEEE Int'l Test Conf., ITC*, 1(1):31.3, 2005.

[2] S. Eichenberger, J. Geuzebroek, C. Hora, B. Kruseman, and A. Majhi. Towards a world without test escapes. *Proc. of IEEE Int'l Test Conf., ITC*, 1(1):1, 2008.

[3] F.Hapke, R.Krenz-Baath, A.Glowatz, J.Schloeffel, H.Hashempour, S.Eichenberger, C.Hora, and D. Adolfsson. Defect-oriented cell-aware atpg and fault simulation for industrial cell libraries and designs. *Iternational Test Conference*, 1(1):1, 2009.

[4] J. Geuzebroek, E. Marinissen, A. Majhi, A. Glowatz, , and F. Hapke. Embedded multi-detect atpg and its effect on the detection of unmodeled defects. *Proc. of IEEE Int'l Test Conf., ITC*, 1(1):1, 2007.

[5] S. K. Goel, N. Devta-Prasanna, and M. Ward. Comparing the effectiveness of deterministic bridge fault and multiple-detect stuck fault patterns for physical bridge defects: A simulation and silicon study. *Iternational Test Conference*, 1(1):1, 2009.

[6] F. Hapke, W. Redemund, J. Schloeffel, R. Krenz-Baath, A. Glowatz, M. Wittke, H. Hashempour, and S. Eichenberger. Defect-oriented cell-internal testing,. *IEEE Int'l Test Conf., ITC*, 1(1):1, 2010.

[7] F. Hapke, M. Reese, J. Rivers, A.Over, V. Ravikumar, W. Redemund, A. Glowatz, J. Schloeffel, and J. Rajski. Cell-aware production test results from a 32-nm notebook processor,. *Proc. of IEEE Int'l Test Conf., ITC*, 1(1):1, 2012.

[8] T.-W. Huang, S.-Y. Yeh, and T.-Y. Ho. A network-flow based pin-count aware routing algorithm for broadcast electrode-addressing ewod chips. *IEEE/ACM ICCAD*, 1(1):425–431, 2010.

[9] K. Lu-Yen, H. Shi-Yu, C. Jia-Liang, and C. Han-Chia. Modeling and testing of intra-cell bridging defects using butterfly structure. *VLSI Design, Automation and Test*, 1(1):1, 2006.

[10] I. Pomeranz and S. Reddy. On n-detection test sets and variable n-detection test sets for transition faults. *VLSI Test Symposium*, 1(1):173–180, 1999.

[11] Z. SUN, A. BOSIO, L. DILILLO, P. GIRARD, A. TODRI, and A. VIRAZEL. Effect-cause intra-cell diagnosis at transistor level. *Int'l Symposium on Quality Electronic Design*, 1(1):1, 2013.

[12] H. Tang, G. Chen, C. Wang, J. Rajski, and I. Pomeranz. Defect aware test pattern. *Design, Automation and Test in Europe Conference and Exihibition*, 1(1):1, 2005.

[13] H. Tang, G. Chen, C. Wang, J. Rajski, I. Pomeranz, and S. Reddy. Defect aware test patterns,. *Proc. of Design, Automation, and Test in Europe (DATE)*, 1(1):450–455, 2005.

[14] T. Xu and K. Chakrabarty. Broadcast electrode-addressing for pin-constrained multi-functional digital microfluidic biochips. *IEEE/ACM DAC,*, 1(1):173–178, 2008.

[15] S.-H. Yeh, J.-W. Chang, T.-W. Huang, and T.-Y. Ho. Voltage-aware chip-level design for reliability-driven pin-constrained ewod chips. *International Conference on Computer-Aided Design*, 1(1):353–360, 2012.