

# **SYNTHESIS OF SOFT ERROR TOLERANT COMBINATIONAL CIRCUITS**

BY

**KHALED ABDEL-KARIM DAUD**

A Thesis Presented to the  
DEANSHIP OF GRADUATE STUDIES

**KING FAHD UNIVERSITY OF PETROLEUM & MINERALS**

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the  
Requirements for the Degree of

**MASTER OF SCIENCE**

In

COMPUTER ENGINEERING

**January, 2012**

**KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS**

DHAHRAN 31261, SAUDI ARABIA

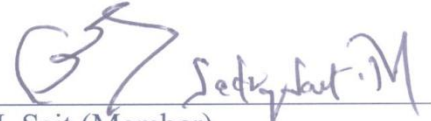
**DEANSHIP OF GRADUATE STUDIES**

This thesis, written by **KHALED DAUD** under the direction of his thesis advisor and approved by his thesis committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE in COMPUTER ENGINEERING**.

**Thesis Committee**



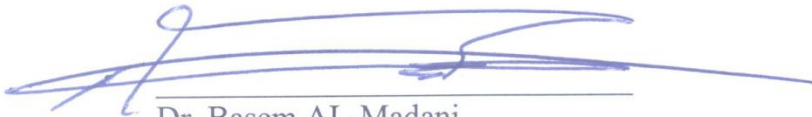
Dr. Aiman El-Maleh (Advisor)



Dr. Sadiq M. Sait (Member)



Dr. Alaaeldin Amin (Member)



Dr. Basem AL-Madani  
(Department Chairman)



Dr. Salam A. Zummo  
(Dean of Graduate Studies)



1/4/12

Date

*Dedicated to My Beloved Parents and  
Brothers*

# ACKNOWLEDGMENT

All praise and thanks are due to Almighty Allah, Most Gracious and Most Merciful, for his immense beneficence and blessings. He bestowed upon me health, knowledge and patience to complete this work. May peace and blessings be upon prophet Muhammad (PBUH), his family and his companions.

Thereafter, acknowledgement is due to the support and facilities provided by the Computer Engineering Department of King Fahd University of Petroleum & Minerals for the completion of this work.

I acknowledge, with deep gratitude and appreciation, the inspiration, encouragement, valuable time and continuous guidance given to me by my thesis advisor, **Dr. Aiman El-Maleh**. I am also grateful to my committee members, **Dr. Sadiq M. Sait**, and **Dr. Alaaeldin Amin** for their constructive guidance and support.

My heartfelt thanks are due to my parents and brothers for their prayers, guidance, and moral support throughout my academic life. My parents' advice, to strive for excellence has made all this work possible.

Last, but not least, thanks to all my colleagues and friends who encourage me a lot in my way to the achievement of this work.

# TABLE OF CONTENTS

<b>ACKNOWLEDGMENT .....</b>	<b>iv</b>
<b>LIST OF TABLES .....</b>	<b>viii</b>
<b>LIST OF FIGURES .....</b>	<b>x</b>
<b>THESIS ABSTRACT (ENGLISH) .....</b>	<b>xii</b>
<b>THESIS ABSTRACT (ARABIC).....</b>	<b>xiv</b>
<b>CHAPTER 1 INTRODUCTION.....</b>	<b>1</b>
1.1 Combinational Circuits.....	3
1.2 Soft Errors in Nano-Scale Circuits .....	3
1.3 Research Motivation.....	12
1.4 Problem Statement and Thesis Objectives .....	12
1.4.1 Problem Statement .....	12
1.4.2 Thesis Objectives .....	13
1.5 Thesis Contributions.....	14
1.6 Thesis Organization.....	15
<b>CHAPTER 2 LITERATURE REVIEW.....</b>	<b>16</b>
2.1 Definitions .....	16
2.1.1 Defects, Faults and Errors .....	16
2.1.2 Defect (or Fault) Models .....	17
2.1.3 Failure Rate .....	18
2.1.4 Reliability .....	18
2.1.5 Fault Tolerance.....	19
2.2 Current Fault Tolerance Methods.....	19
2.2.1 Hardware Redundancy Techniques .....	20
2.2.2 Synthesis-Based Fault Tolerance Techniques .....	29
2.2.3 Physical Characteristics Based Fault Tolerance Techniques.....	31

<b>CHAPTER 3 TWO-LEVEL LOGIC SYNTHESIS FOR SOFT ERROR TOLERANCE.....</b>	<b>33</b>
3.1 Extracting Sub-Circuits .....	34
3.2 Adding Controllability Don't Care Conditions .....	36
3.3 Proposed Design Level Approach.....	38
3.3.1 Finding Care Minterms' Probabilities .....	38
3.3.2 Two-Level Fail Rate Estimator.....	39
3.3.3 Finding Covering Cubes .....	44
3.3.4 Adding Redundant Cubes .....	51
3.3.5 Duplicate One Cube Phase .....	52
3.3.6 Algorithm 1: Two-Level Circuit Synthesis .....	53
3.3.7 Complexity of Algorithm 1.....	55
3.4 Conclusion.....	56
<b>CHAPTER 4 MULTI-LEVEL LOGIC SYNTHESIS FOR SOFT ERROR TOLERANCE .....</b>	<b>58</b>
4.1 Fast Extraction for Area Optimization .....	59
4.2 Fast Extraction for Reliability .....	61
4.3 Algorithm 2: Enhancing Area for Reliability Optimized Two-Level Circuits...	69
4.4 Framework of Enhancing Fault Tolerance Using Algorithms 1 and 2 .....	70
4.5 Conclusion.....	72
<b>CHAPTER 5 SIMULATION ENVIRONMENT AND FRAMEWORK .....</b>	<b>73</b>
5.1 Measuring Combinational Circuit Reliability .....	74
5.2 The Simulation Framework of Reliability Evaluator .....	74
5.3 Assumptions .....	76
5.4 Fault Model and Fault Injection Mechanism.....	76
<b>CHAPTER 6 RESULTS, DISCUSSION AND FINDINGS .....</b>	<b>78</b>
6.1 Experiments .....	78
6.2 Calculating the Area Overhead .....	83

6.3 Algorithm 1 Results.....	83
6.3.1 Case Study: Bench1 Benchmark .....	84
6.3.2 Other Benchmarks Results .....	88
6.3.3 Aggregated Results and Conclusions .....	101
6.4 Algorithm 2 Results.....	108
6.5 Overall Results Using Algorithm 1 and Algorithm 2.....	114
6.6 Comparison Results.....	118
6.7 Proposed Technique with TMR.....	121
<b>CHAPTER 7 CONCLUSION AND FUTURE WORK.....</b>	<b>123</b>
7.1 Summary of the Contributions .....	125
7.2 Future Work.....	126
<b>REFERENCES.....</b>	<b>127</b>
<b>VITA.....</b>	<b>133</b>

## LIST OF TABLES

Table 3.1: Summary of failure rate estimator example.....	42
Table 3.2: Selected cubes covering for example 1. ....	45
Table 3.3: Cubes covering for example 2. ....	50
Table 4.1: Double-cube divisors along with their bases for $f = ade + ag + bcde +$ bcg.....	59
Table 4.2: Failure rate for two implementations of $f_2 = ab + ac + de$ for different minterms probabilities. ....	66
Table 4.3: Failure rate for two implementations of $x = abf + adef + cde$ for different minterms probabilities. ....	68
Table 6.1: primary inputs, primary outputs and area for benchmark circuits.....	79
Table 6.2: Size of gates.....	83
Table 6.3: Failure rate results for MCNC combinational benchmark circuits - Injecting 1 fault.....	104
Table 6.4: Failure rate results for MCNC combinational benchmark circuits - Injecting 5 faults. ....	105
Table 6.5: Failure rate results for MCNC combinational benchmark circuits - Injecting 10 faults. ....	106
Table 6.6: Area overhead for combinational benchmark circuits.....	107
Table 6.7: Overall averaged failure rate results for benchmark circuits.....	108
Table 6.8: Failure rate results for combinational benchmark circuits using Algorithm 2 - Injecting 1 fault.....	110
Table 6.9: Failure rate results for combinational benchmark circuits using Algorithm 2 - Injecting 5 faults. ....	111
Table 6.10: Failure rate results for combinational benchmark circuits using Algorithm 2 - Injecting 10 faults. ....	112
Table 6.11: Area reduction for combinational benchmark circuits using Algorithm 2. ....	113
Table 6.12: Overall averaged failure rate overhead for benchmark circuits using Algorithm 2.....	114



Table 6.13: Overall results of failure rate for benchmark circuits using fixed number of faults 1, 5 and 10. ....	116
Table 6.14: Overall results of failure rate for benchmark circuits using fixed number of faults 1, 5 and 10 at the original version. ....	117
Table 6.15: Area overhead for combinational benchmark circuits using proposed technique compared to original circuits. ....	118
Table 6.16: Failure rate results for Ranking-based Don't Care comparison. ....	120
Table 6.17: Area overhead of Ranking-based DC compared to Algorithm 1 versions. ....	120
Table 6.18: Failure rate results for TMR experiments. ....	122
Table 6.19: Area overhead for TMR experiments. ....	122

## LIST OF FIGURES

Figure 1.1: Upsets hitting combinational and sequential logic.....	4
Figure 1.2: Single Event Upset (SEU) effect in an SRAM memory cell. ....	5
Figure 1.3: NMOS transistor hit by ion particle. ....	6
Figure 1.4: Soft error generation due to alpha particle strike. ....	8
Figure 1.5: Logical masking. ....	9
Figure 1.6: Electrical masking [9]. ....	10
Figure 1.7: Latching window masking [9].....	11
Figure 2.1: Von Neumann’s logic for 2-input NAND gate with $N = 4$ . ....	22
Figure 2.2: A Triple Modular Redundant (TMR) structure.....	23
Figure 2.3: Quadded logic example: (a) original circuit, (b) quadded logic circuit. ....	25
Figure 2.4: Partial error masking scheme as proposed in [33]. ....	26
Figure 2.5: HICC module [34].....	28
Figure 3.1: Extracting sub-circuits’ algorithm.....	35
Figure 3.2: Example of 1-level window. ....	37
Figure 3.3: Failure rate estimator example. ....	40
Figure 3.4: Two-level failure rate estimator algorithm.....	43
Figure 3.5: K-map for example 1.....	45
Figure 3.6: Different implementations for example 1. ....	47
Figure 3.7: Faulty wires in example 1 for minterm 1110 for second implementation. ....	47
Figure 3.8: K-map for example 2.....	50
Figure 3.9: Duplication of one cube off-phase. ....	52
Figure 3.10: Duplication of one cube on-phase.....	53
Figure 3.11: Algorithm 1- Two-level tool to enhance fault tolerance. ....	54
Figure 4.1: Fast Extraction algorithm for area optimization [50]. ....	61
Figure 4.2: Single-cube extraction example: (a) $f_1 = abc + abd + abe + kg$ , (b) $t$ $= ab$ , $f_1 = tc + td + te + kg$ . ....	62
Figure 4.3: Double-cube extraction example: (a) $f_2 = ab + ac + de$ , (b) $t = b+c$ , $f_2 = at + de$ . ....	65

Figure 4.4: Double-cube extraction with complement example: (a) $f_3 = ac + bc + abd$ , (b) $t = a + b$ , $f_3 = tc + t'd$ .....	67
Figure 4.5: Algorithm 2. ....	70
Figure 4.6: Framework of Algorithm 1 and Algorithm 2. ....	71
Figure 5.1: Fault injection mechanism.....	77
Figure 6.1: Framework of experiments.....	82
Figure 6.2: Failure rate vs Faults for bench1. ....	86
Figure 6.3: Size of bench1 experiments.....	87
Figure 6.4: Failure rate vs Faults for m3.....	88
Figure 6.5: Failure rate vs Faults for test1. ....	89
Figure 6.6: Failure rate vs Faults for test4. ....	90
Figure 6.7: Failure rate vs Faults for ex1010.....	91
Figure 6.8: Failure rate vs Faults for misex3. ....	92
Figure 6.9: Failure rate vs Faults for exp.....	93
Figure 6.10: Failure rate vs Faults for apex4. ....	94
Figure 6.11: Failure rate vs Faults for duke2.....	95
Figure 6.12: Failure rate vs Faults for spla. ....	96
Figure 6.13: Failure rate vs Faults for cps. ....	97
Figure 6.14: Failure rate vs Faults for table3.....	98
Figure 6.15: Failure rate vs Faults for table5.....	99
Figure 6.16: Failure rate vs Faults for apex3. ....	100

# THESIS ABSTRACT (ENGLISH)

**NAME:** KHALED ABDEL-KARIM DAUD

**TITLE:** SYNTHESIS OF SOFT ERROR TOLERANT  
COMBINATIONAL CIRCUITS

**MAJOR FIELD:** COMPUTER ENGINEERING

**DATE OF DEGREE:** JANUARY 2011

Due to current technology scaling trends, digital designs are becoming more sensitive to radiation-induced particle hits resulting from radioactivity decay and cosmic rays. A low-energy particle can flip the output of a gate, resulting in a soft error if it's propagated to a circuit output. Thus, soft error tolerance has become an important criterion in digital system design. This work is directed to analyze, model and design combinational circuits for soft error tolerance. A simulation based method to reduce the soft error failure rate in combinational logic circuits is proposed. This method maximizes the probability of logical masking when a soft error occurs. This is done by extracting sub-circuits from the original multi-level circuit and then re-synthesizing each extracted sub-circuit to increase fault masking. After that, the re-synthesized sub-circuits are merged back to the original circuit. Therefore, the overall reliability of the original circuit will be enhanced as well. We present a two-level synthesis scheme to maximize soft error masking that is applied on each extracted sub-circuit. This scheme provides a heuristic that first finds the best irredundant set of cubes to cover an extracted sub-circuit minterms. This cover maximizes fault masking against single fault especially for minterms with high

probability of occurrence. Then, an extra number of cubes can be added as redundant cubes to the cover such that they have a significant effect on maximizing error masking. Reliability driven fast extraction is also proposed to enhance area overhead of synthesized two-level circuits. Experimental results on some MCNC combinational benchmarks show that on average, a failure rate reduction of 52% is achieved compared to the original circuits. The area overhead on average is found to be 61% of the original circuit.

**Keywords:** *fault tolerance, soft errors, transient faults, single event upset (SEU), single event transients (SET), nano technology, robust system design, circuit reliability, combinational circuits, soft error rate.*

# THESIS ABSTRACT (ARABIC)

## ملخص الرسالة

الاسم: خالد عبد الكريم داود  
عنوان الرسالة: تصميم دوائر توافقية ذات وقاية من الاخطاء اللحظية  
التخصص: هندسة الحاسب الآلي  
تاريخ التخرج: يناير 2012 م

مع تقدم تقنية النانو، أصبحت الانظمة الرقمية أكثر قابلية للأخطاء المستحثة بواسطة الذرات الأيونية. حيث أنه يمكن لجسيم ذو طاقة منخفضة أن يغير قيمة بوابة منطقية معينة، فينتج عن ذلك ما يسمى الخطأ اللحظي إذا انتقل هذا التغيير إلى مخارج الدائرة. وبالتالي، أصبحت الوقاية من الأخطاء اللحظية معياراً مهماً في تصميم الأنظمة الرقمية. هذا البحث موجه لتحليل وتمثيل وتصميم الدوائر التوافقية وذلك لزيادة وقايتها من الأخطاء اللحظية ذات الطابع المستحدث أيونياً. اقترحنا طريقة تقلل نسبة الخطأ في الدوائر التوافقية بالنسبة للأخطاء اللحظية، هذه الطريقة مبنية على زيادة احتمالية إلغاء أثر الخطأ اللحظي بالاعتماد على التركيب المنطقي للدائرة. في هذه الطريقة يتم استخراج دوائر أصغر من الدائرة الأصلية، و إعادة تصميم هذه الدوائر المستخرجة بحيث تكون أكثر وقاية من الأخطاء اللحظية. ومن ثم يتم إرجاع الدوائر المعاد تصميمها إلى الدائرة الأصلية. التقنية المقترحة لإعادة تصميم الدوائر المستخرجة تقوم على إيجاد أفضل مجموعة من الحدود الغير مكررة لتغطية مدخلات الدائرة المستخرجة. هذه المجموعة توفر أقصى قدر ممكن من الحماية ضد خطأ لحظي واحد خاصة بالنسبة للمدخلات ذات الاحتمالية العالية للحدوث. بعد ذلك، يتم إضافة حدود مكررة للمجموعة بحيث يكون هناك تحسين معتبر في الحماية. اقترحنا تقنية أخرى مبنية على خوارزمية الاستخراج السريع التي يمكن استخدامها لتقليل المساحة الزائدة الناتجة عن تطبيق الطريقة السابقة على الدوائر المستخرجة. نتائج التجارب على الدوائر التوافقية MCNC تبين أنه تحقق تقليل في معدل الخطأ بنسبة 52% بالمقارنة مع الدوائر الأصلية. كما أنه وجد أن تكلفة المساحة الإضافية ما يقارب 61% من مساحة الدوائر الأصلية.

الكلمات الرئيسية: التسامح في الأخطاء أو العيوب، الأخطاء اللينة، العيوب اللحظية، الحدث الأحادي القالب، الحدث الأحادي اللحظي، تقنية النانو، تصميم الأنظمة القوية، اعتمادية الدوائر، الدوائر التوافقية، معدل الأخطاء اللحظية.

# Chapter 1

## INTRODUCTION

Reliability with respect to soft errors has become a critical issue in digital circuits. In the past few decades, CMOS technology has reached high scaling advancement. This advancement is consistent with Moore's law [1] which states that the number of transistors that can be placed in a chip doubles every 18 months. As CMOS technology is improving and reaching the nanometer scale, quantum mechanical effects come into the picture generating many challenges for additional scaling of CMOS devices. This has motivated researchers to investigate new technologies for circuit design. Circuits and devices based on nanotechnology-based fabrication are expected to offer extra density and performance that takes electronic circuits to next higher integration level. According to [2], nanoelectronics can operate at very high frequencies (of the order of THz) and achieve very high densities ( $10^{12}$  devices per  $\text{cm}^2$ ). Several researchers have presented novel successful nanoelectronic devices. These devices include carbon nano-tubes(CNT) [3], silicon nano-wires (NW) [4-5], and quantum dot cells [6].

Nanoscale devices are limited by several characteristics. The most dominant characteristics are the devices' higher defect rates and increased susceptibility to soft errors. These limiting characteristics are due to two sources [7]:

- One source is the inherent randomness and imprecision in the bottom-up manufacturing process, which results in a large number of defective devices during the fabrication process.
- The second source is the reduced noise tolerance of these devices which is responsible for inducing device malfunctions by external influences like EMI (electromagnetic interference), thermal perturbations and cosmic radiations.

In general, errors can be categorized as either permanent or transient errors. Permanent (hard) errors may occur during manufacturing process or during the lifetime of a device. Transient (soft) errors can arise due to multiple sources like high-energy particles, coupling, power supply noise, leakage and temporal circuit variations. The transient error can last for one or many clock cycles. Both types of errors affect the circuit reliability if they aren't tolerated. Reliability of a circuit can be defined as the ability to function properly despite the existence of such errors. It's required to improve the tolerance against permanent and transient errors in order to enhance circuit reliability.

In this work, we address the fault tolerance of transient (soft) errors for combinational circuits. In the following sections, we will introduce the combinational circuit model as well as the nature of transient faults. After that, a glance will be given about this work motivation, objectives and contributions. Then we will finish by thesis organization.



## 1.1 Combinational Circuits

Logic circuits for digital systems may be combinational or sequential. A combinational circuit consists of logic gates (AND, OR, NOR, NAND... etc) whose outputs at any time are determined by combining the values of the applied inputs using logic operations. A combinational circuit performs an operation that can be specified logically by a set of Boolean expressions. Sequential circuits employ storage elements in addition to using logic gates. In combinational circuits, the output is a pure function of the present input only. This is in contrast to sequential logic, where the output depends not only on present input but on the history of the input as well.

A combinational circuit consists of input variables, output variables, logic gates and interconnections. The interconnected logic gates accept signals from the inputs and generate signals at the output. The  $n$  input variables come from the environment of the circuit, and the  $m$  output variables are available for use by the environment. Each input and output variable exists physically as a binary signal that represents logic 1 or logic 0.

## 1.2 Soft Errors in Nano-Scale Circuits

Transient faults (SET/SEU) are mainly caused by cosmic-ray neutrons or alpha particles through the materials of ICs. They can hit either in the combinational logic or flip flops of a sequential circuit block. If it happens in the combinational logic, it will result in a Single Event Transient (SET) fault. On the other hand, if it happens in the memory cell itself, it will result in a Single Event Upset (SEU) fault. Both of SET and SEU faults cause a major implication and should be treated properly.

Figure 1.1 shows a typical structure for most sequential circuits. In this figure, the first latch releases data to combinational logic at the rising or falling clock edge, and then combinational logic performs operation. Latch 2 stores result outputs at rising or falling clock edge.

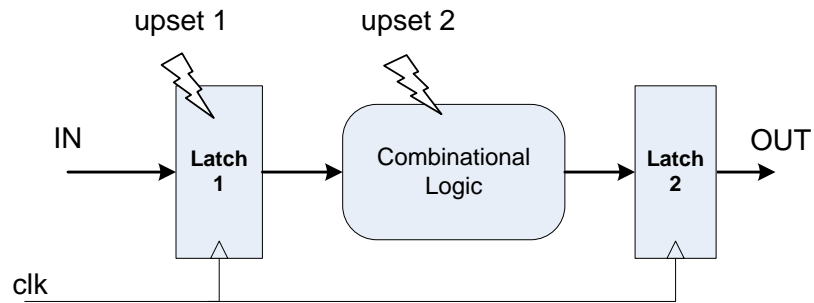


Figure 1.1: Upsets hitting combinational and sequential logic.

When a charged particle strikes a sensitive region in a memory cell, such as a drain in an OFF transistor, a transient current pulse is generated that can cause a bit flip in the memory cell. A memory cell stores two states either logic 0 or 1 values. In each state, two transistors are ON and two are OFF. Figure 1.2 illustrates how an energetic particle can reverse the state of transistors in the circuit, which results in a bit-flipping.

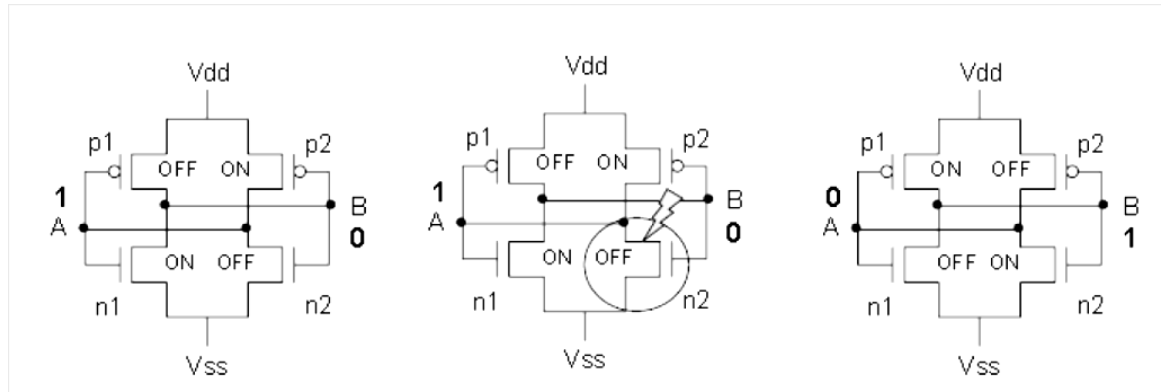


Figure 1.2: Single Event Upset (SEU) effect in an SRAM memory cell.

A single event transient (SET) occurs when a charged particle hits the combinational logic, resulting in a transient current pulse. This can change the logic level of a gate. If this transient has enough width and magnitude, it can result in an erroneous value to be latched. Once it is latched, a single event transient becomes a single event upset (SEU). However, there are certain conditions that are to be met for this transient to propagate and be latched by the memory element. These conditions are discussed later.

It is worth to mention that a single SET can produce multiple transient current pulses at the output. This is due to the logic fan-out in the circuit. Hence, SETs can produce multiple SEUs in the memory elements.

For more details on how a transient soft error can change the state of transistor, consider the NMOS transistor shown in Figure 1.3 (a). The transistor is assumed to be in the ON state. During normal operation, a current will flow from the drain to the source that makes the transistor ON. If an alpha particle strikes the drain of the NMOS transistor, it loses its energy as it travels along the path inside the semiconductor material.

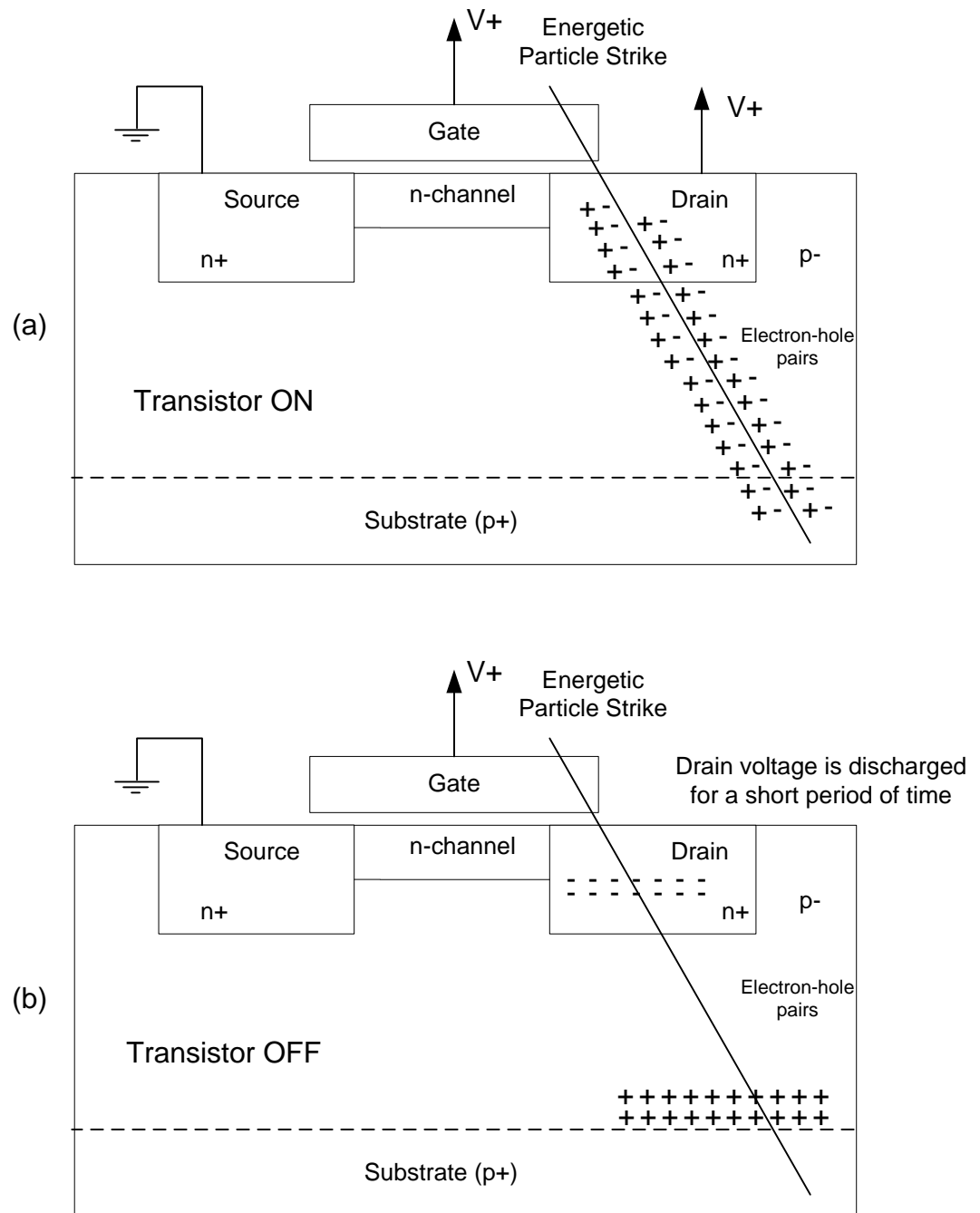


Figure 1.3: NMOS transistor hit by ion particle.

In this period, the particle ionizes the material around it, which results in the generation of electron-hole pairs. If there is no voltage applied to the drain of transistor, electrons and holes are recombined. However, since the transistor is in the ON state, electrons and holes are separated by the voltage applied on the drain. Consequently, the holes are collected by the (p+) substrate and the electrons are collected by the drain. This results in a prompt component of current at the drain in shape of negative pulse; this is shown in Figure 1.3 (b). If this prompt current has a high enough charge, this will lead to discharging the voltage at the drain for a very short period of time in order of 100 to 200 picoseconds [8]. Hence, the state of transistor is changed to OFF in that period of time.

In the previous generations of CMOS technologies, the sizes of CMOS transistors were large enough to neglect the effect of the resulting prompted current. However, with device dimensions shrinking to nanometer scale, SET and SEU faults are no longer considered a small attenuation. Instead, they will be considered as normal circuit signals. Therefore, tolerance of soft and transient errors is no longer limited to specific applications like aerospace applications, and they can no longer be ignored.

In order to give insight into how a particle strike hits in the combinational logic can generate wrong values stored in latches, consider the circuit in Figure 1.4. Assume that the inputs of the AND gate A1 are  $A = '0'$  and  $B = '1'$ . During normal conditions, the output of A1 is '0'. Also the output of gate A2 is '0'. Therefore, the memory element latches a logic value of '0' during normal operation. Let us consider a particle strike at the output node of gate A1 which results in the change of logic level at the output of A1. The particle strike in this case is assumed to have sufficient pulse width and magnitude so that it propagates through the gate A2 and gets latched by the memory element. Now we have

a wrong value of '1' latched by the memory element instead of '0'. This type of error is called soft error.

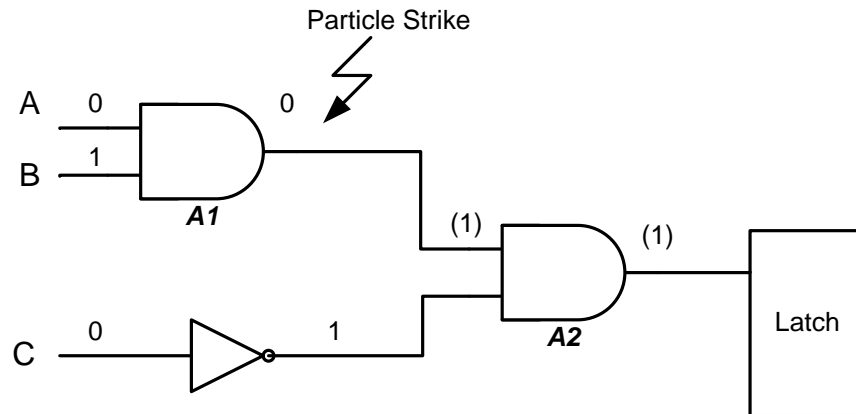


Figure 1.4: Soft error generation due to alpha particle strike.

Although the incident alpha particles cause voltage transients, these transients must propagate through a certain path to get latched and result in soft errors. This path is called critical path. The following are three types of masking that shield the SEUs from propagating.

- Logical Masking
- Electrical Masking
- Latching window masking

## Logical Masking

Logical masking prevents the SET from propagation from fault location to primary outputs of circuit because of path gate inputs that stop logical transition of the gate's output. As shown in Figure 1.5, there is a particle strike at the output of the A1 gate which results in a wrong logic of '1' instead of logic '0'. This wrong value is one of the inputs of the A2 gate.

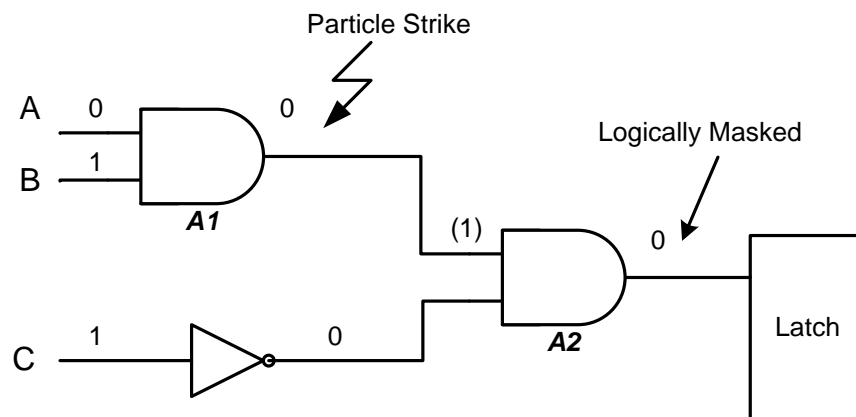


Figure 1.5: Logical masking.

When one of the inputs of A2 gate is tied to logic '0', the output of A2 gate is always logic '0' irrespective of the other input. Therefore, this input of A2 gate is called controlling input. The transient caused by the alpha particle strike is logically masked. Hence a correct value is latched by the following memory element.

## Electrical Masking

Electrical masking attenuates or completely masks the SET signal due to electrical properties of gates. The voltage transient caused by the particle strike is attenuated as it propagates through a series of gates. The transient gets attenuated to an extent where it is ignored by the following memory element.

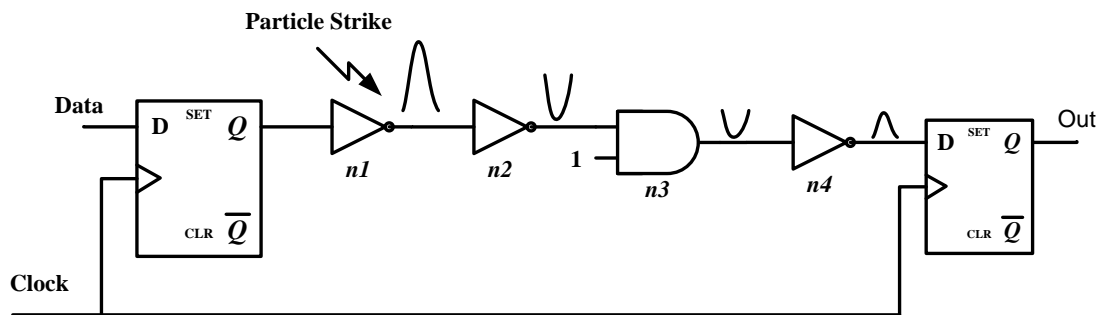


Figure 1.6: Electrical masking [9].

As shown in Figure 1.6, the voltage pulse generated at the output of the gate  $n1$  attenuates as it passes through gates  $n2$ ,  $n3$  and  $n4$ . The attenuation is due to the parasitic capacitance of succeeding gates. The pulse with duration more than the gate delay attenuates as it propagates [9].



## Latching Window Masking

In latching window masking, if a SET doesn't arrive "on time" then it will be masked; this depends on hold and setup times of the target memory element. This is a timing related masking technique. For a voltage transient to get latched by a memory element, the pulse should be available exactly at the latching window. The transient is masked if it arrives before or after the latching window. As shown in Figure 1.7, the value of 'out' changes only when the glitch is available at the latching window. In all the other cases, the output is error free.

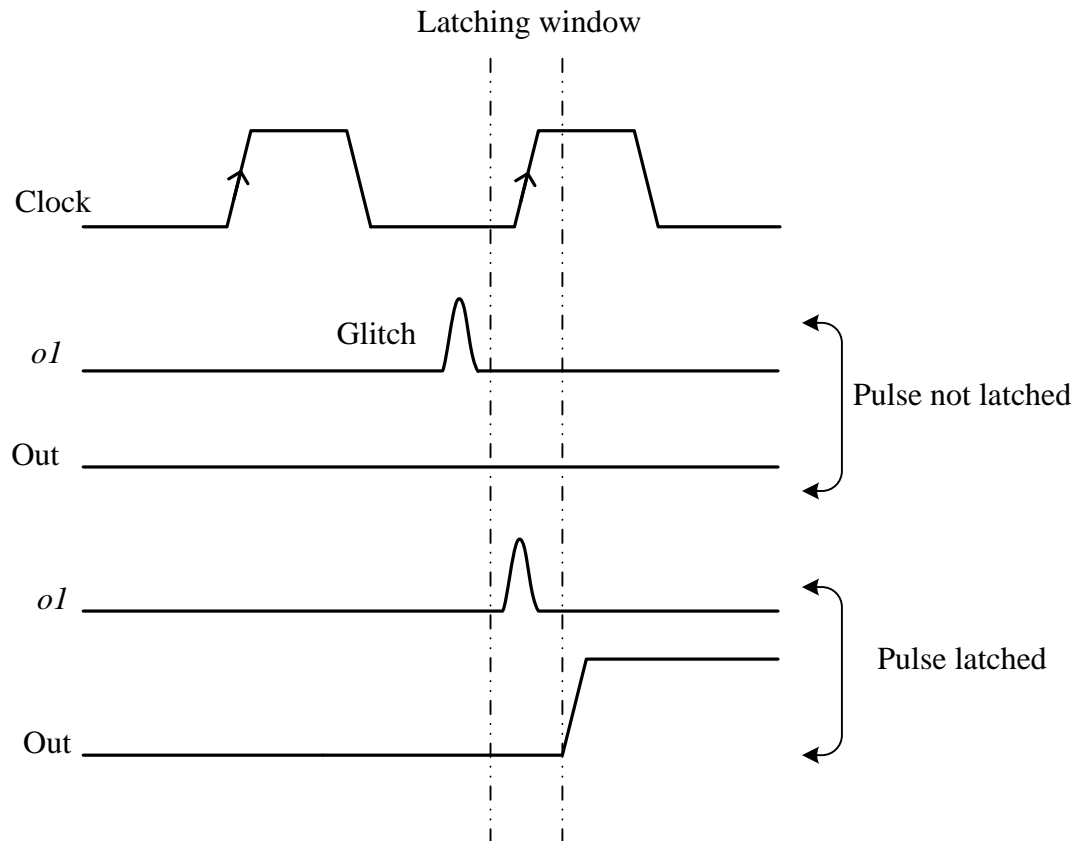


Figure 1.7: Latching window masking [9].

These three mechanisms can prevent some SETs from being latched. However, the continuous scaling trends limit the effectiveness of the electrical and latching window masking properties.

### **1.3 Research Motivation**

With the advancement in technology reaching 0.35 microns and below, systems became more susceptible to soft errors. Soft errors induced by ion particles are no longer considered a small attenuation and can no longer be neglected. Therefore, techniques are needed to tackle soft faults in both combinational and sequential circuits. Many techniques were proposed by researchers to enhance circuit reliability against soft errors. In this work, a method is proposed to enhance combinational circuits' reliability. This is achieved by synthesizing the circuit in order to maximize the probability of logical masking when a soft error occurs.

### **1.4 Problem Statement and Thesis Objectives**

Problem statement is specified as well as the thesis main contributions in the following sub-sections.

#### **1.4.1 Problem Statement**

The general problem statement is: *Given a combinational circuit, we would like to increase its reliability against transient errors with minimum area overhead.*

### 1.4.2 Thesis Objectives

The objective of this work is to investigate the design of soft error tolerant combinational circuits based on maximizing the probability of logical masking when a soft error occurs. This is done by extracting small sub-circuits from the original circuit. Then for each extracted sub-circuit, the probabilities of its input vectors to occur are computed. Next, developed tool is applied on each extracted sub-circuit to produce a new two-level sub-circuit. Logical masking for one fault is maximized in the new two-level sub-circuit. Finally, the new synthesized sub-circuits are merged back to the original circuit. The main work objectives can be summarized as:

- Design a two-level tool that maximizes fault masking against the occurrence of a single fault in a circuit. Maximizing masking is based on the probabilities of sub-circuit input vectors to occur.
- Design a multi-level tool that reduces the area overhead result after applying the two-level tool. Applying this multi-level tool is supposed to have low impact on the reliability enhancement achieved.
- Design a framework to be used in order to apply the previous two techniques on the original circuit.

## 1.5 Thesis Contributions

To achieve the thesis objectives, the contributions of this thesis can be summarized as follows:

- Implement a tool for extracting sub-circuits from an original multi-level circuit, finding inputs don't cares, and the probability of care minterms for each extracted sub-circuit.
- Implement a tool for computing soft error reliability for combinational circuits based on Monte Carlo simulation [10]. The objective of this tool is to find the failure rate of a combinational circuit as more faults are observed in the circuit.
- Develop and implement an algorithm to enhance reliability of combinational circuits based on enhancing reliability of each individual sub-circuit. A two-level synthesis heuristic is proposed and applied on each extracted sub-circuit that attempts to find the best cover that maximizes fault masking while keeping area overhead minimum.
- Develop and implement a multi-level tool that reduces the area overhead resulting after applying the two-level tool. This should have a low impact on fault masking achieved by the two-level tool.
- Evaluate the proposed approaches in terms of failure rate and area overhead.

## 1.6 Thesis Organization

The thesis is organized into the following chapters. Chapter 2 starts with a background about the current fault tolerance methods. An exploration of different approaches to solve this problem at different levels of abstraction will be presented.

Chapter 3 discusses in detail the proposed two-level technique, namely Algorithm 1, which enhances the logical masking of a circuit against a single fault. In Chapter 4, detailed discussion is presented on the proposed multi-level technique, namely Algorithm 2. Chapter 5 describes the simulation framework used to evaluate algorithms proposed in Chapter 3 and Chapter 4. Chapter 6 presents the experimental results along with discussion. The thesis finally concludes in Chapter 7, where the proposed solutions are summarized, with a list of some potential improvements as a future work.

## Chapter 2

# LITERATURE REVIEW

In this chapter, definitions regarding fault tolerance are reviewed. Then, a survey of the current methods to tolerate faults in combinational circuits is presented.

### 2.1 Definitions

In the next few subsections, some terms are defined. Those terms are used to describe when electronic systems fail. Throughout the thesis, many of those terms are used.

#### 2.1.1 Defects, Faults and Errors

The definitions of defects, errors and faults terms as defined in [11] are presented.

##### Defects

A defect in the electronic system is the unintended difference between the implemented hardware and its intended design. Some typical defects in VLSI chips are:

- Process Defects - missing contact windows, parasitic transistors, oxide breakdown.
- Material Defects – bulk defects (cracks, crystal imperfections), surface impurities.
- Age Defects – dielectric breakdown, electro-migration etc.
- Package Defects – contact degradation, seal leaks.

Defects occur either during manufacturing or during the use of devices. Repeated occurrence of the same defect indicates the need for improvement in the manufacturing process or the design of the device.

### **Faults**

A representation of a “defect” at the abstracted function level is called a fault. The difference between a defect and a fault is rather subtle. They are the imperfections in the hardware and function respectively.

### **Errors**

A wrong output signal produced by a defective system (or circuit) is called an error. An error is an effect whose cause is some “defect”. Fabrication defects, fabrication errors and physical failures are collectively termed as physical faults [11]. According to their stability in time, physical faults can be classified as:

- *Permanent faults*: They are those which are always present after their occurrence.
- *Intermittent faults*: They are those which exist only during some intervals.
- *Transient faults*: They are one-time occurrence (also known as Single Event Upsets (SEUs) or Single-Event Transients (SETs)) which are caused by a temporary change in some environment factor e.g., due to  $\alpha$ -particle radiation etc.

## **2.1.2 Defect (or Fault) Models**

In order to simulate the effect of faults in the circuit, many fault models have been proposed. Those models are presented in different levels of abstraction. Examples of such

models are: stuck-at defect (or fault) model, *stuck open and stuck-short defect (or fault) model*, *bridging defect (or fault) model* and *crosspoint defect (or fault) model*. In this work, a single stuck-at fault model is used. The interested reader can refer to [11] for other fault models. In the stuck-at fault model, a fixed value of 0 or 1 is assigned to a single line in the circuit. A single line can be an output or an input of a gate. Single stuck-at faults are the most famous because they are simple and model many types of defects. They are also known as stuck-at-1 and stuck-at-0 faults.

### **2.1.3 Failure Rate**

Failure rate of a system represents the regularity at which a system fails. It can be defined as the total number of failures within an item population, divided by the total time expended by that population, during a particular measurements interval under stated conditions [12]. In this study, failure rate is redefined as: the total number of experiments in which a circuit fails divided by the total number of performed experiments.

Mainly, in semiconductor industry, the Failures In Time (FIT) rate is used to represent the failure rate of a system. It represents the expected number of failures in one billion ( $10^9$ ) hours of device operation. In other words, 1000 devices for 1 million hours, or 1 million devices for 1000 hours each, or some other combination [13-15].

### **2.1.4 Reliability**

The reliability of a system can be defined as the ability to perform specified function under stated conditions [16]. For hardware systems, the most common way of evaluating reliability is to apply a probabilistic reliability function  $R(t)$  that gives the probability that



a system is working correctly between time 0 and time t, given certain conditions and correct behavior at time 0.

### **2.1.5 Fault Tolerance**

A system is fault-tolerant if its programs can be properly executed despite the occurrence of faults [17]. The objective of fault-tolerance is either to mask, or to recover from, faults once they have been detected [18].

## **2.2 Current Fault Tolerance Methods**

In general, two types of techniques are used to reduce soft error failure rate. The first one, *fault avoidance*, in which the defective modules are identified and are replaced by other redundant modules through configuration. The second one, *fault tolerance*, fault-tolerant techniques are based on the concept of adding redundancy in order to mask faulty behavior due to faults, defects or errors. Fault-tolerant techniques attempt to maximize the probabilities of the three masking mechanisms, namely, logical, electrical and latching window masking.

Avoidance mechanisms presented in the literature generally exploit fabrication process (device-level) to reduce charge collection [19-23]. Error detection circuits[24-25] can be used to monitor the outputs of the circuit when an error occurs. If it detects an error, the system recovers through rollback and retry to prevent the failures.

The fault-tolerant techniques work on circuit-level or higher levels of abstractions to achieve soft error rate (SER) improvement. Fault-tolerant techniques for combinational

circuits can be classified into three major categories: *hardware redundancy*, *synthesis-based*, and *physical characteristics based* techniques.

Hardware redundancy methods are based on adding redundant hardware. Multiple modules are used to represent the same function in order to maximize masking of errors. Multiple copies of either the entire circuit or part of the circuit are used as redundant hardware.

In the synthesis-based techniques, the combinational circuit is restructured in order to maximize masking properties of the circuit. Logical masking is the main masking property to be maximized.

The physical characteristics based techniques attempt to reduce SER based on the physical characteristics to maximize the electrical masking.

In this section, a survey of the current fault-tolerant methods to tolerate SEU/SET in combinational circuits are discussed.

## **2.2.1 Hardware Redundancy Techniques**

### **Von Neumann's Multiplexing**

Designing of reliable systems by using redundant unreliable components was initiated by John von Neumann in the 1950s [26]. He proposed a multiplexing technique in which a processing unit is replaced by multiplexed units. Two stages are used to implement a unit: the *executive* stage and the *restorative* stage. The *executive* stage represents the basic function of the unit, while the *restorative* stage is used to correct some of the faulty values of the gates caused by errors in the *executive* stage. In the *executive* stage, unit is

replaced by  $N$  multiplexed units which have  $N$  copies of every input and output of the unit. The inputs randomly pair to feed the  $N$  units. Consider the case when the processing unit is a single 2-input NAND gate, with  $N=4$ , Von Neumann multiplexing is shown in Figure 2.1. The unit  $U$  represents a random permutation of the input signals. The two inputs of each NAND gate are selected randomly from the first and second inputs  $X$  and  $Y$  respectively. The restorative stage is made using the same technique as in the executive stage. However, the outputs of the executive stage are duplicated and used as inputs for the restorative stage. Note that, this approach will invert the result if it's used only once, thus, two steps are required. By defining some critical level  $\Delta$  such that  $0 < \Delta < 1/2$ , if the number of lines carrying a positive state (logic 1) is larger than  $(1 - \Delta)N$ , he considers it as a positive state of the bundle, if it was less than  $\Delta$ , he interprets it as negative state (logic 0). In cases where the number of positive state lines does not meet either of these criteria, then the output is not decided, and so a fault will occur.

Giving a probability of failure  $\varepsilon$  for each gate, Von Neumann's structure requires a large amount of redundancy and a low error rate for individual gates. For deep logic with a gate failure probability  $\varepsilon = 0.01$  and  $N = 100$ , it is shown in [27] that a circuit failure probability in the order of  $10^{-6}$  can be obtained. This required amount of redundancy is huge and is considered impractical. In order to reduce this large amount of redundancy, the works in [28-29] combine NAND multiplexing with reconfiguration.

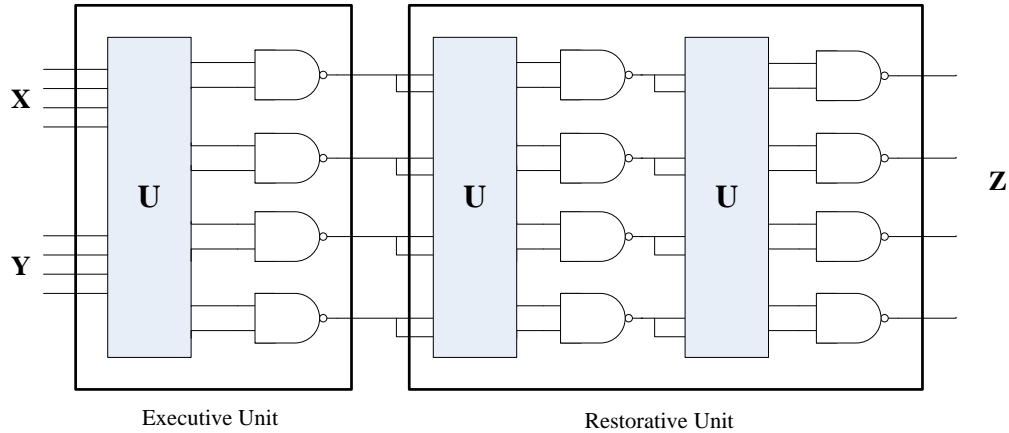


Figure 2.1: Von Neumann's logic for 2-input NAND gate with  $N = 4$ .

### Triple Modular Redundancy (TMR)

Triple Modular Redundancy is one of the most well-known techniques to tolerate soft/hard errors in combinational circuits [16]. It's a special case of the NMR system. An NMR system (also known as  $M$ -of- $N$  system) is a system that consists of  $N$  modules and needs at least  $M$  of them for proper operation. TMR is a system where  $M=2$  and  $N=3$ , which consists of three functionally identical copies of the original circuit that feed a 2-out-of-3 majority voter as shown in Figure 2.2. If 2 modules out of 3 produce expected correct results, then the majority of the modules produces correct results, and so the error in the third module will be masked. However, TMR suffers from high overhead in terms of area and power (more than 200%).

In such a structure,  $M=2$  and  $N=3$  and voter selects the majority vote. If a single voter is used, that voter becomes a critical point of failure and the reliability of the TMR structure is limited by that of the final arbitration unit (i.e., voter), which makes the approach difficult in the context of highly integrated nano-systems [20]. Despite this

limitation, TMR is heavily used in practice especially when single faults are needed to be protected. Even in the case of multiple faults, some of these faults could be masked due to electrical and logical masking in each module.

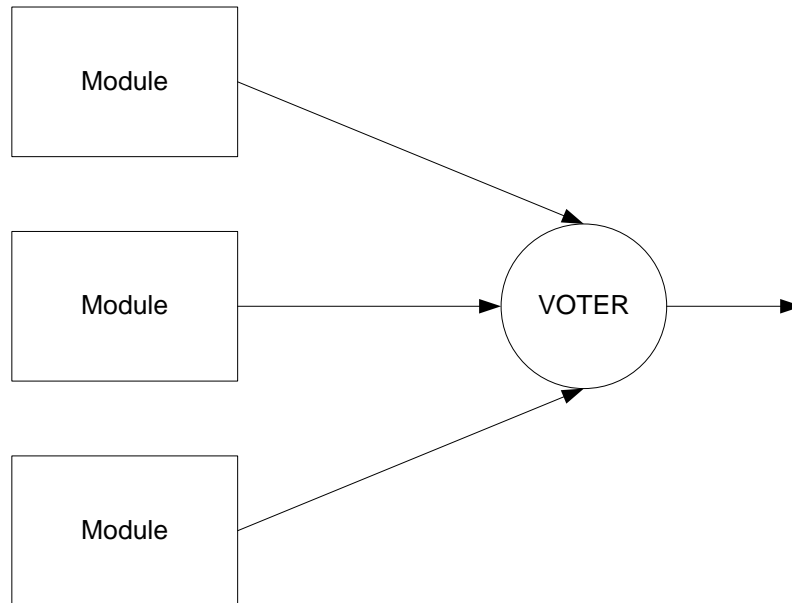


Figure 2.2: A Triple Modular Redundant (TMR) structure.

### **Interwoven Redundant Logic and Quadded Logic**

Pierce[30] suggested another scheme called interwoven redundant logic. This scheme considers two types of faults  $0 \rightarrow 1$  and  $1 \rightarrow 0$  faults. The error correction mechanism in interwoven redundant logic depends on asymmetries in the effects of these two types of binary errors. The effect of a fault depends on the value of the input and the type of gate. Consider a NAND gate, for an instance. If the value of one of the inputs is 0 while it should be 1, the output of NAND gate will be 1 regardless of the values of other inputs. In this case the output will be stuck at 1. On the other hand, if an input value is 1 while it

should be 0, the output will depend on other inputs and the output will not be stuck. The type of faults that cause the output to be stuck is considered as critical; the other type is subcritical in the sense that its occurrence alone does not cause an output error. Hence, alternating layers of NAND (or NOR) gates can correct errors by switching them from critical to subcritical.

Quadded logic [31-32] is an ad hoc configuration of the interwoven redundant logic. It requires four times as many circuits, interconnected in a systematic way, and it corrects errors and performs the desired computation at the same time. A quadded circuit implementation based on NAND gates replaces each NAND gate with a group of four NAND gates, each of which has twice as many inputs as the one it replaces. The four outputs of each group are divided into two sets of outputs, each providing inputs to two gates in a succeeding stage. The interconnections in a quadded circuit are eight times as many as those used in the non-redundant form. In a quadded circuit, a single critical error ( $1 \rightarrow 0$ ) is correctable after passing through two stages of logic and a single sub-critical error ( $0 \rightarrow 1$ ) will be corrected after passing a single stage. In quadded logic, it must be guaranteed that the interconnect pattern at the output of a stage differ from the interconnect patterns of any of its input variables. While quadded logic guarantees tolerance of most single errors, errors occurring at the last two stages of logic may not be corrected. Figure 2.3 shows an example of a quadded logic circuit.

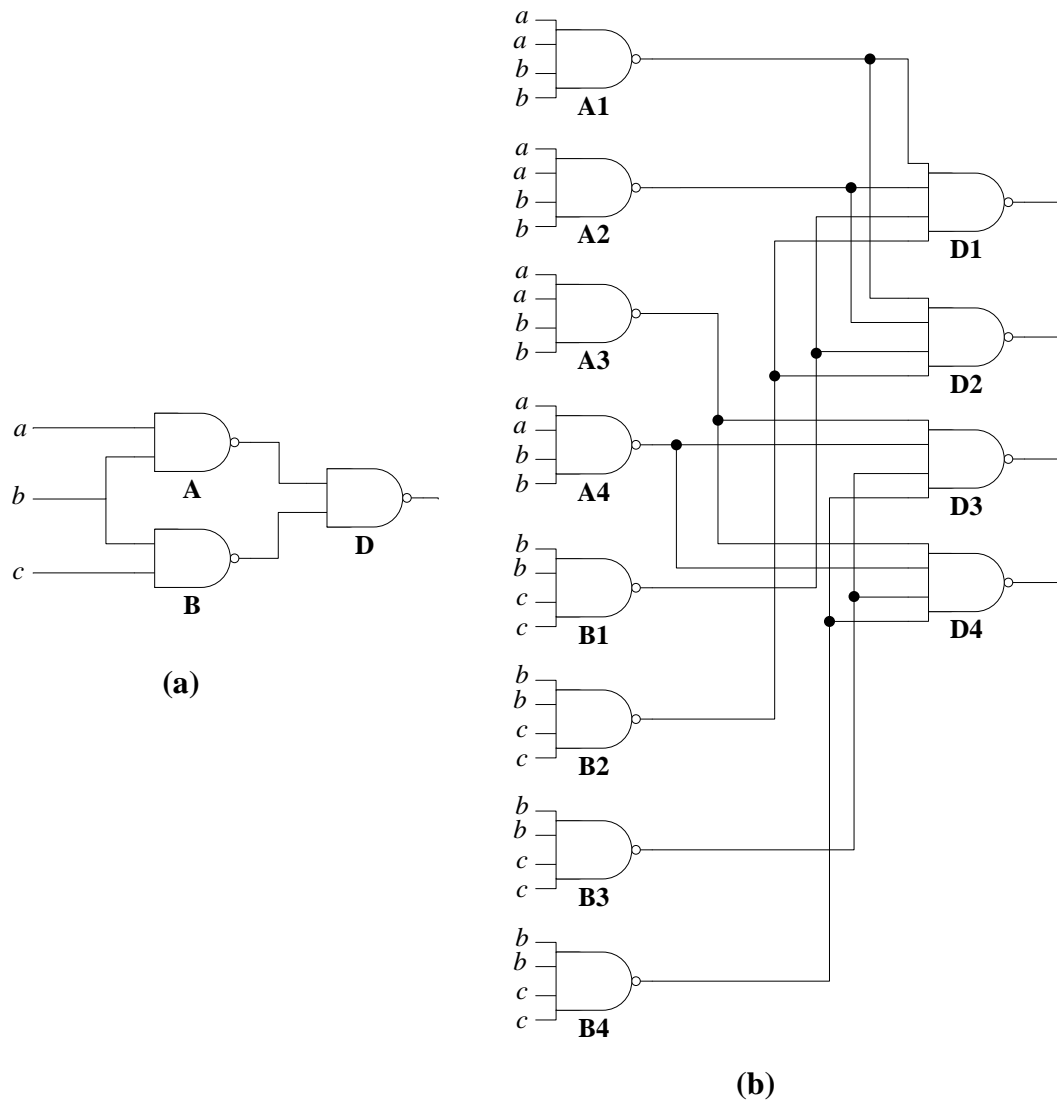


Figure 2.3: Quadded logic example: (a) original circuit, (b) quadded logic circuit.

### Partial Error Masking Scheme Based on TMR

In [33], a partial error masking scheme is proposed based on TMR shown in Figure 2.4. It targets the nodes with the highest soft error susceptibility. Two reduction heuristics are used to reduce soft error failure rate, namely, cluster sharing reduction and dominant value reduction. Instead of triplicating the whole logic as in TMR, only the nodes with highest soft error susceptibility are triplicated, the rest of nodes are clustered and are shared among the triplicated logic. The dominant value reduction heuristic exploits the fact that the logic 0 and logic 1 soft error susceptibility of certain primary outputs is highly skewed. Such outputs are identified and the triplication is replaced by duplication. The 2-out-of-3 majority is replaced by AND (OR) logic.

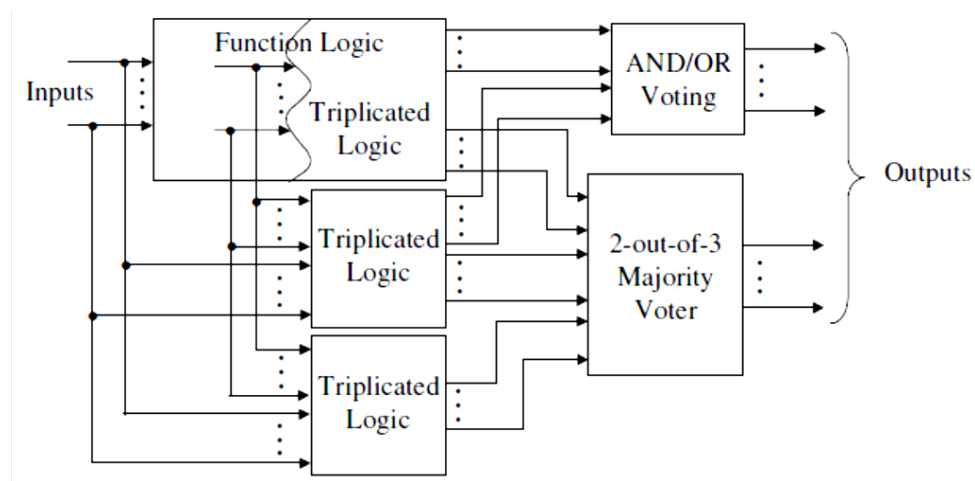


Figure 2.4: Partial error masking scheme as proposed in [33].



### **Fault Tolerant Based on History Index of Correct Computation**

A more recent technique based on TMR is proposed in [34]. A history index of correct computation (HICC) module is used to select the correct result. Instead of using merely majority voting to transmit results, HICC module uses the history indices of redundant units to transmit the correct computation. It represents a measure of a hardware unit's reliability. The most reliable unit is the unit with the highest history index. The computations of other redundant units that implement the same function are ignored.

Figure 2.5 shows an example that demonstrates the concept of the HICC module. In the figure, an ALU module is triplicated as units *A*, *B*, and *C*. The *result selector* decides the unit with the correct result based on stored history index of each unit. The unit with the highest index is considered to be the most reliable unit, and its result is transmitted. When all units have the same history index value, a bitwise majority voting is used to decide the result. After that, the history index of each unit is incremented by 1 if its result is identical to the result of majority; otherwise it's decremented by 1. The HICC logic is distributed within the modules themselves. Hence, unreliable modules are identified simultaneously in real time and are ignored.

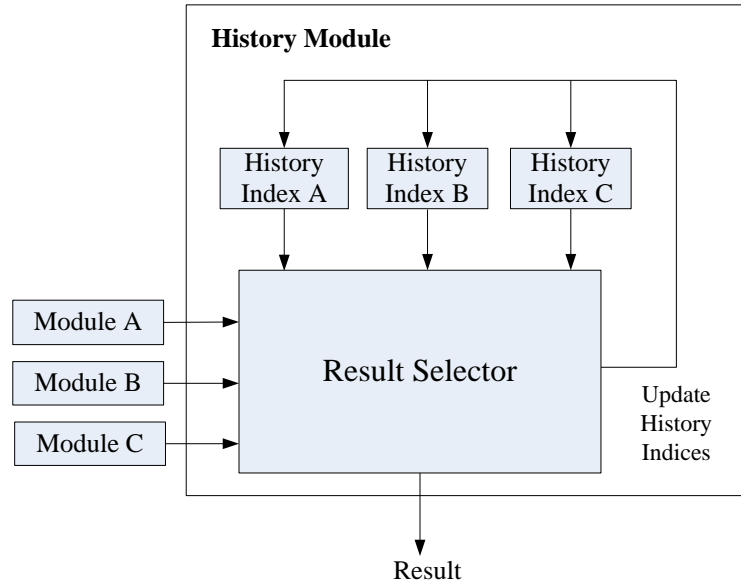


Figure 2.5: HICC module [34].

### Defect-Tolerant $N^2$ Transistor Structure

In [35], a defect tolerant technique that adds redundancy at the transistor level of the circuit is proposed. The  $N^2$  structure is a generalization of the quadded-transistor structure. In the quadded-transistor structure, each transistor,  $A$ , is replaced by a structure that implements either the logic function  $(AA) + (AA)$  or the logic function  $(A+A)(A+A)$ . In such structure, any single transistor defect is tolerated. However, in the  $N^2$  structure,  $N$  blocks are connected in series such that each block contains  $N$  parallel transistors. If number of defects is less than or equal to  $(N-1)$ ,  $N^2$  structure guarantees the tolerance of all those defects. It was shown that this technique achieves higher defect tolerance compared to gate level based techniques such like quadded logic and TMR.

## 2.2.2 Synthesis-Based Fault Tolerance Techniques

### Localized Circuit Restructuring Tolerant Technique

In [36], logic masking of errors is increased by taking the advantage of conditions already present in the circuit, such as observability don't cares. Two techniques are used to improve reliability: don't care-based resynthesis and local rewriting. In the first method, high-impact nodes are identified. A node has high impact if many observable faults flow through it. High-impact nodes are used to select areas of the circuit for restructuring, in which a vulnerable node is replicated by adding a single gate. Local rewriting is also used to optimize small sub-circuits to obtain overall area improvements.

In the first method, logic masking is increased by getting benefit from redundancy already present in the circuit. Covering relationships are identified among existing nodes. Given two nodes  $x$  and  $y$ ,  $x$  covers  $y$  if and only if  $x$  is 1 or a don't-care whenever  $y$  is 1. Node  $x$  is defined to be an *anti-cover* of node  $y$  if  $y$  is 1 or a don't-care whenever  $x$  is 1. An *impact* measure is defined and calculated for each node. High impact nodes are selected to be redesigned. For a high-impact node  $x$ , other nodes which node  $x$  covers or anti-covers are found. Given a node  $y$  covered by  $x$ , redundant logic is added by transforming node  $x$  into  $OR(x,y)$ . Similarly, if  $x$  is an anti-cover of  $y$ , node  $x$  is transformed into  $AND(x,y)$ . When  $x$  is replaced by  $AND(x,y)$  all errors that flip  $x$  from 0 to 1 will be masked. Likewise, all 1 to 0 errors will be masked by  $OR(x,y)$ . Covering relation can be implied from nodes' signatures. For a node  $n$ , a signature is defined as the sequence of logic values observed at  $n$  due to a sequence of  $K$  input vectors. Thus, the signature partially specifies the Boolean function of  $n$ . For example, suppose  $x$  has a

signature  $sig(x) = 11000$  and  $sig(y) = 11001$ . By definition,  $x$  is an anti-cover of  $y$ , because  $y$  is 1 whenever  $x$  is 1. Therefore,  $x$  can be replaced by  $AND(x,y)$ . And so, all 0-to-1 flips at the third and fourth inputs will be masked. However, if  $y$  is replaced by  $OR(x,y)$  then all 1-to-0 flips of the first two bits will be masked. Since signatures do not fully capture Boolean functions, SAT solver is used to verify replication of nodes.

### **Reliability-Driven Don't Care Assignment Method**

In [37], two algorithms are proposed to improve input error resilience. They focus on input error due to propagated failures from previous blocks. Both algorithms determine 0/1 assignments for the most critical DC terms. Consider the correct input vector for a circuit is 0100, if a fault happens that fails the third input, the 0110 vector will be applied to the logic circuit. If the implementation is identical for these two vectors, then the error will be masked. If 0110 is a don't care, then the assignment of this minterm to either 0 or 1 will determine the masking of an error on the third input of the 0100 vector. Given a circuit with a set of don't care minterms, the output after applying proposed algorithms is the circuit with new on-set minterms, new off-set minterms, and new don't cares set. The new on-set contains the original on-minterms plus don't cares assigned to on-minterms. The new off-set contains the original off-minterms plus don't cares assigned to off-minterms. Finally, the new don't care set contains the don't cares that are left unassigned. Assignment process uses Hamming-distance metrics; don't care minterm is assigned to 1 if the number of neighboring 1's is greater than 0's, if the neighboring 0's are greater than 1's, it will be assigned to 0. If 0's and 1's are neutral, then don't care minterm is left unassigned.

## **Redundancy Addition and Removal Technique**

In [38], a framework is proposed based on redundancy addition and removal for soft error rate (SER) reduction. It performs a series of wire addition and removal by searching for redundant wires in the circuit. It will go through an iterative process trying to keep wires/gates with higher masking impact and to remove wires/gates with higher error impact; this will be guided using some metrics. The masking impact takes into account the three masking mechanisms.

### **2.2.3 Physical Characteristics Based Fault Tolerance Techniques**

Many methods found in literature attempt to reduce SER based on the physical characteristics to maximize the electrical masking and latching-window masking. Gate resizing strategy [39] reduces SER by modifying the W/L ratios of transistors in gates. To achieve significant improvement in SER, potentially large overheads in area, delay, and power are introduced. In [40], a related method is introduced, which uses optimal assignments of gate sizes, threshold voltages, supply voltages, and output capacitive loads to get better results while keeping overheads smaller. Nevertheless, the design complexity is increased in this method in addition to the possibility of making circuit hard to optimize at physical design. Another scheme [41] focuses on the selection of flip-flop from a given set. It increases the probability of preventing faulty transients from being registered by selectively lengthening latching-windows associated with flipflops, but it doesn't consider logical masking and electrical masking. A hybrid approach [42] combines flip-flop selection with gate resizing to achieve SER improvement.

A more recent technique is presented in [43] that attempts to increase electrical masking. In this method the impact of using reliability-aware logic synthesis to reduce both the pulse width and the drain area of a circuit is analyzed. The idea here is to replace highly vulnerable cells with alternative cells or logical functions to reduce overall vulnerability of a circuit. The pulse width and drain area are used in this study as the reliability metrics to rank cells. The strategy is as follows: circuits are synthesized to a given cell library, then, the frequently used and highly vulnerable cells are identified; those identified cells are removed from library and are replaced with alternative implementations. Thus, an improved cell library is created.

In [44], a circuit simplification method is proposed for error tolerant applications. In some applications such as images, video, audio and graphics many faulty versions of a chip can be used. In this work, the original combinational circuit is given with a defined error threshold, and the minimum area simplified circuit version is derived such that the error it produces is within the given threshold.

## Chapter 3

# **TWO-LEVEL LOGIC SYNTHESSES FOR SOFT ERROR TOLERANCE**

In this chapter, we will introduce a novel technique to increase combinational circuit reliability and hence fault tolerance. The technique is based on maximizing the probability of logical masking when a soft error occurs. This is done by extracting small sub-circuits from the original circuit. Then, for each extracted sub-circuit, the probabilities of its care input vectors are computed. Next, two-level logic synthesis for soft error tolerance is applied on each extracted sub-circuit. Logical masking for single fault is maximized in the synthesized two-level sub-circuit. Finally, the new synthesized sub-circuits are merged back to the original circuit

A two-level synthesis scheme is presented to maximize soft error masking. This scheme provides a heuristic that first finds the best irredundant set of cubes to cover an extracted sub-circuit minterms. This set of cubes is constructed such that fault masking for single fault is maximized especially for minterms with high probability of occurrence.

Then, an extra number of cubes can be added as redundant cubes to the cover such that they have significant improvement on maximizing error masking.

For each extracted sub-circuit, either the on-phase or the off-phase is synthesized and used to implement the circuit. The phase which has minterms with higher probability of occurrence is implemented.

According to our knowledge, none of the previous techniques has presented the introduction of maximizing fault masking according to covering cubes for extracted sub-circuits.

### **3.1 Extracting Sub-Circuits**

The proposed method involves synthesizing small extracted sub-circuits from the original circuit, and by enhancing the reliability of such extracted sub-circuits, the overall reliability of the original circuit will be enhanced as well. In this section, we will explain how sub-circuits are extracted from an original multi-level circuit.

The input of this phase is a multi-level circuit, and the output will be a set of smaller sub-circuits, such that each extracted sub-circuit has a single output, and a maximum number of inputs  $M$ . The value of  $M$  should not exceed 15 in order to synthesize the sub-circuit in a feasible time. As the extracted sub-circuits will be synthesized individually, each extracted sub-circuit should have one output, maximum  $M$  inputs, and it should have no fan-outs to other circuits, since the structure of the sub-circuit will be changed after synthesis.



Extraction of sub-circuits is done as follows: starting from primary outputs of the original circuit, keep adding gates into sub-circuits, level by level until reaching the maximum number of inputs for a sub-circuit  $M$ , and without adding fan-out gates. Each extracted sub-circuit is then converted into a two-level circuit in the PLA format using the SIS tool [45]. The algorithm of extracting sub-circuits is shown in Figure 3.1.

---

**inputs:** original circuit, maximum number of inputs  $M$ .

1. Initialize  $E$  as the collection of extracted sub-circuits.
2. Add all primary outputs of the original circuit into  $List1$ .
3. **while** ( $List1$  is not empty ) {
  - 3.1.  $O$  = get the next element in  $List1$ .
  - 3.2. Initialize  $e$  as the structure of the new extracted sub-circuit.
  - 3.3. Starting from  $O$ , go back in reverse order in the original circuit, and add gates to  $e$  under the following conditions:
    - Prune branching when reaching a fan-out gate or primary input.
    - Terminate adding gates to  $e$  when reaching  $M$ .
  - 3.4. Add extracted sub-circuit  $e$  to  $E$ .
  - 3.5. For every input  $i$  of  $e$  sub-circuit:
    - **if** ( $i$  isn't included in  $List1$  AND  $i$  isn't a primary input )  
 Add  $i$  to the end of  $List1$ .     //  $i$  will be the output of a new sub-circuit
- } // end while
4. Return ( $E$ ).

---

Figure 3.1: Extracting sub-circuits' algorithm.

### 3.2 Adding Controllability Don't Care Conditions

To increase flexibility, and to optimize area as well, controllability don't care conditions (CDCs) for each sub-circuit can be used. CDCs are defined as the input patterns that are impossible to occur at the inputs of a circuit. A windowing method with simulation is developed. The reader can refer to [46-48] for other techniques to compute don't cares.

To find CDCs, a window is constructed for each extracted sub-circuit. It contains the sub-circuit gates in addition to  $L$  levels of gates. The additional gates are added by going back traversal starting from the inputs of the sub-circuits. Figure 3.2 shows an example of 1-level window. The shaded nodes in the figure represent the extracted sub-circuit, and the un-shaded nodes represent the additional nodes added for one level.

CDCs conditions are found using simulation as follows: for each sub-circuit, a window circuit is extracted from the original circuit. Then, all possible input patterns of the window circuit are simulated using a developed simulator. For each input pattern, the observed input vector on the inputs of sub-circuit is added to the care set for this sub-circuit. Don't care set can then be easily found by removing care set vectors from all possible input vectors for the sub-circuit. If the number of primary inputs of the original circuits is small; i.e., less than 25, the window can be constructed starting from the output of sub-circuit, and then back traversing until reaching the primary inputs of the original circuit. In such case, we can apply all possible input combinations on the inputs of the window, and we can get all possible CDCs of sub-circuit of interest. However, if the

number of primary inputs is large, we can construct a window by adding levels so that we have a feasible number of inputs.

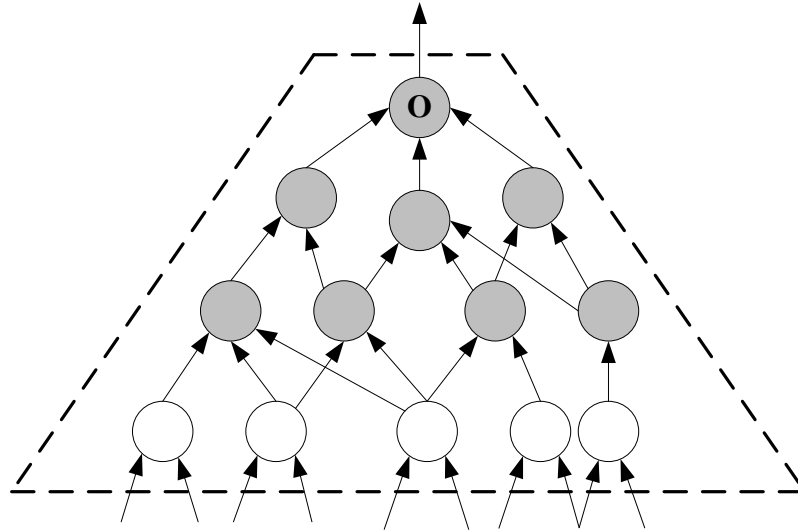


Figure 3.2: Example of 1-level window.

A key observation about CDCs in this case, is that they are compatible, i.e., they can be derived for each sub-circuit and used independently. Optimizing a sub-circuit using the derived CDCs won't have any effect on other extracted sub-circuits. After finding CDCs for each extracted circuit, they can be added to the pla file of the sub-circuit.

### 3.3 Proposed Design Level Approach

In order to reduce the soft error effect, the proposed two-level design level approach protects the highly probable input patterns by selecting the best fault masking cover and then by adding extra redundant cubes. By keeping other input patterns with low probability without adding redundancy, the area overhead is kept minimal.

#### 3.3.1 Finding Care Minterms' Probabilities

The proposed method is based on care minterms' probabilities. An assignment of  $n$  Boolean variables is called a minterm. The positive and negative minterms correspond to the assignment, for which a function takes a value of 1 and 0, respectively. The union of positive and negative minterms is called the *care minterms*.

The set of care minterms for each extracted sub-circuit is already found in the stage of finding CDCs; i.e., all possible input patterns for a sub-circuit except the CDCs. The probability of each care minterm to occur for each extracted sub-circuit is calculated using simulation. A set of all care minterms combinations for all extracted sub-circuits is constructed, and then a set of random input patterns are applied on the primary inputs of the original circuit, and for each care minterm, the probability for this minterm to occur is calculated. This is done by dividing the number of times a minterm occurs in simulations over the number of used input patterns. The input patterns can be the all possible combinations if the number of primary inputs of original circuit is feasible; otherwise, a number of random simulations are used.

### 3.3.2 Two-Level Fail Rate Estimator

In this section, a tool is described which estimates the failure rate of a two-level circuit for a single fault based on the fault model explained in Chapter 5. This tool will be used later in the proposed scheme. This tool goes over all minterms; on and off minterms, and calculates the number of faulty wires for each minterm. The algorithm for estimating the failure rate of a two-level circuit is shown in Figure 3.4 . A faulty wire is a wire such that if its value changes due to an error, its faulty value will not be masked. The way to find the number of faulty wires for on-minterms is different than that for off-minterms as will be explained in the coming example. The estimated failure rate for a two-level circuit for a single fault can be estimated according to the following equation:

$$\text{Estimated Failure Rate} = \frac{\sum_{m \in \text{on}}(p_m * f_m) + \sum_{m \in \text{off}}(p_m * f_m)}{\text{Total Wires} * 2} \quad (1)$$

$p_m$  is the probability of a minterm  $m$  and  $f_m$  is the number of faulty wires for minterm  $m$ . The division over 2 is due to the number of stuck-at faults that may occur on a wire.

Consider a circuit with the following cover: (101-, 1-01, and --10). The implementation of this cover is shown in Figure 3.3. Suppose the on-minterms are: (0010, 1011, and 1010) and off-minterms are: (1111, and 0100).

The 0010 minterm is covered by only one cube (--10); the faulty wires considering wires' numbering in Figure 3.3 are as follows: wire 4 is faulty since the minterm is covered by only one cube (cube 3), that is, if an error hits it will not be masked by any other inputs of the OR gate, since all other inputs will be zero. All input wires to cube 3 are faulty, namely wires 11, 12 and 18 since an error in any of these wires will change the

value of the AND gate and make it zero. Since no other cube covers this minterm, the zero value of AND will not be masked, and it will propagate to output causing a false output. The fan-out wires 15 and 17 are also faulty, because an error in any of them will fail an input value for the cube. The output of the OR gate (wire 1) is always faulty, since there is no masking for it. Hence, the total number of faulty wires for 0010 minterm is 7.

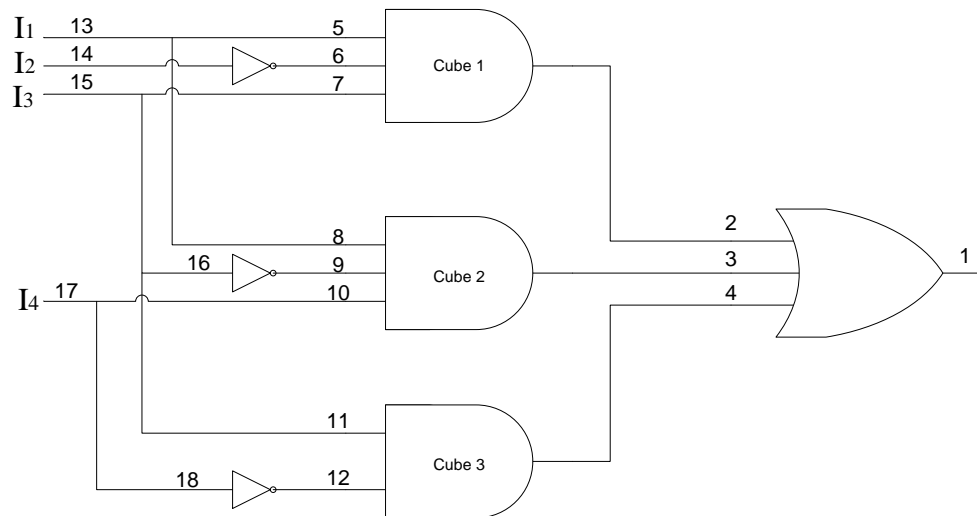


Figure 3.3: Failure rate estimator example.

The 1011 minterm is also covered by one cube only (101-). The wires 1, 2, 5, 6, 7 and 14 are faulty wires as explained before. The fan-out wire 13 is also faulty. If we consider the fan-out wire 15; the correct value on it is 1. If an error hits and changed its value to 0 then the value of cube 1 will be zero. However, the second input of cube 2 will be one, and so, all inputs of cube 2 are ones. Therefore, the fault in wire 15 is masked, because the change in its value makes another cube (cube 2) to be 1. The total number of faulty wires for 1011 minterm is 7.

The 1010 minterm is covered by two cubes, namely 101- and --10 cubes. In this case, none of the OR gate inputs will be faulty; since an error in any of those cubes output will be masked by the other one. Also, none of the two cubes inputs are faulty. However, there are faulty wires due to fan-outs. The fan-out wire 15 is faulty in this case, since an error will cause both values of cubes to be zero. The total number of faulty wires for 1010 minterm is 2.

The way to find the faulty wires for off-minterms is different. For off-minterms, the output should be zero, and no covering is considered in this case. If an error hits in any of the OR inputs, it causes one of the inputs to be logically one; and it can't be masked. However, fault masking can be considered at the inputs of each AND gate. If an input of the AND gate is zero, then an error in any of the other inputs will be masked, since the value of AND gate will be zero.

Considering the off-minterm 1111, the output and input wires of the OR gate are always faulty and so the wires 1, 2, 3 and 4 are faulty. Consider cube 1, since the second input is zero, then the other inputs of the AND gate are not faulty. However, the second input wires are faulty, namely, wires 6 and 14. For cube 2, wires 9, 16 and 15 are faulty. In cube 3, wires 12, 18 and 17 are faulty. The total number of faulty wires for 1111 minterm is 12.

For the off-minterm 0100, the wires 1, 2, 3 and 4 are faulty. For cube 1, all of the three inputs are zeros and hence none of the input wires of this cube is faulty, because an error in any input will be masked by the two other zero inputs of the AND gate. In cube 2, two inputs are zeros, the first and the third inputs and hence none of the input wires of this

cube is faulty, because an error in any input will be masked by at least one other zero input of the AND gate. For cube 3, only the first input is zero, and so the wires 11 and 15 are faulty. The total number of faulty wires for 0100 minterm is 6.

Table 3.1 shows summary of minterms faulty wires along with their probabilities. The estimated failure rate for this circuit can be calculated as follows:

$$f = \frac{(0.1 * 7 + 0.2 * 7 + 0.5 * 2) + (0.07 * 12 + 0.13 * 6)}{18 * 2} = 0.131$$

Table 3.1: Summary of failure rate estimator example.

	Minterm	Probability	Number of faulty wires
On Minterms	0010	0.1	7
	1011	0.2	7
	1010	0.5	2
Off Minterms	1111	0.07	12
	0100	0.13	6



---

**inputs:** cover  $S$ , on-set and off-set minterms with their probabilities.

1.  $TotalFailureRate = 0$ .
2. For every  $m$  in the on-set minterms:
  - 2.1. // Find  $F_m$ ; the number of faulty wires.
    - 2.1.1.  $F_m = 1$ . // Due to the output wire of circuit.
    - 2.1.2.  $C =$  set of cubes  $\in S$  that cover the on-minterm  $m$ .
    - 2.1.3. **if** ( $|C| = 1$ ) {
      - $I_C =$  number of cube inputs.
      - $N_C =$  number of inverted cube inputs.
      - $F_m = F_m + 2 * N_C + I_C - N_C$ . // Faults due to inputs of cube.
      - $F_m = F_m + 1$ . // Due to the output of the cube.
      - For every fan-out wire that feeds the cube, add 1 to  $F_m$  if it's faulty.
      - // A fan-out wire is faulty if its error doesn't change another cube value to 1.
  - else** { // the cover  $S$  has multiple cubes.
    - $V =$  common inputs between cubes in  $C$ .
    - For every input  $i$  in  $V$ :
      - if** ( $i$  is inverted)  $F_m = F_m + 2$ .
      - if** ( $i$  is faulty fan-out)  $F_m = F_m + 1$ .
- 2.2.  $P_m =$  probability of  $m$  to occur.
- 2.3.  $TotalFailRate = TotalFailRate + F_m * P_m$ .

- 3. For every  $m$  in the off-set minterms:
- 3.1. // Find  $F_m$ ; the number of faulty wires.
  - 3.1.1.  $F_m = |S| + 1$ . // Due to the inputs and output of the OR gate.
  - 3.1.2.  $FanOuts = \{ \}$ . // processed fanouts.
  - 3.1.3. For every cube  $c_i$  in the cover  $S$ :
    - $Z =$  number of zero inputs of  $c_i$  after applying minterm  $m$ .
    - if** ( $Z = 1$ ) {
      - $l =$  the zero input in cube  $c_i$ .
      - if** ( $l$  is not fan-out)
        - { **if** ( $l$  is inverted)  $F_m = F_m + 2$ . **else**  $F_m = F_m + 1$ . }
      - else** { //  $l$  is fan-out.
        - $F_m = F_m + 1$ . // Due to zero input  $l$ .
        - if** ( $l$  is not inverted )
          - { **if** ( $l \notin FanOuts$ )  $F_m = F_m + 1$ , add  $l$  to  $FanOuts$ . }
        - else** { //  $l$  is inverted.
          - if** ( $l$  is fanning out from primary input only)  $k = 1$ .
          - else if** ( $l$  is fanning out from inverter output only)  $k = 2$ .
          - else if** ( $l$  is fanning out from primary input and inverter)  $k = 3$ .
          - if** ( $k = 1$ )  $F_m = F_m + 1$ . // Due to inverter input.
          - if** ( $l \notin FanOuts$ ) {  $F_m = F_m + k$ , add  $l$  to  $FanOuts$ . }
- 3.2.  $P_m =$  probability of  $m$  to occur.
- 3.3.  $TotalFailRate = TotalFailRate + F_m * P_m$ .
- 4. Return ( $TotalFailureRate$ ).

---

Figure 3.4: Two-level failure rate estimator algorithm.

### 3.3.3 Finding Covering Cubes

In this section, the method for selecting covering cubes for each extracted sub-circuit is presented. The inputs of this phase are a pla file that contains the minterms of a sub-circuit in addition to the CDCs, and a file that contains the probabilities of sub-circuit care minterms. All prime cubes can be found using ESPRESSO tool. The goal here is to find a set of cubes that covers all minterms and gives better reliability with minimal area overhead. Finding an exact solution is computationally complex and infeasible especially if the number of generated prime implicants is large. However, a heuristic is developed which is a greedy algorithm.

The idea here is to maximize error masking for minterms with high probability of occurrence. Selecting the phase to implement either on or off is very important. It depends on the percentage of minterms each phase covers. The phase with higher probability of occurrence is better to implement. The probability of a phase can be calculated easily by summing the probability of its minterms. Selecting covering cubes is done for the phase of circuit with the higher probability.

Some cubes are mandatory to be selected, which are the essential cubes, so these cubes (if exist) will be selected first. After that, all other cubes will be assigned a weight; this weight represents how much a cube is improving error masking. The cube with the highest weight is selected if it covers new minterms, otherwise the next highest weight cube will be selected and so on.

To illustrate the idea, consider the K-map shown in Figure 3.5 for an extracted circuit with four inputs, where 1 represents an on-set minterm and x represents a don't care

input. The function of the circuit has four primary implicants, namely -100, 11-0, 1-10, and 101-. We need at least two cubes (implicants) to cover all on-minterms. Following are some possible covers that can be used:  $[(-100, 1-10), (11-0, 1-10)]$ . Table 3.2 shows the covering for a set of cubes to the on-minterms. The first column contains the on-minterms, the second column contains the probability for each minterm to occur at the inputs of the circuit, and the last three columns show the minterms covered by each cube.

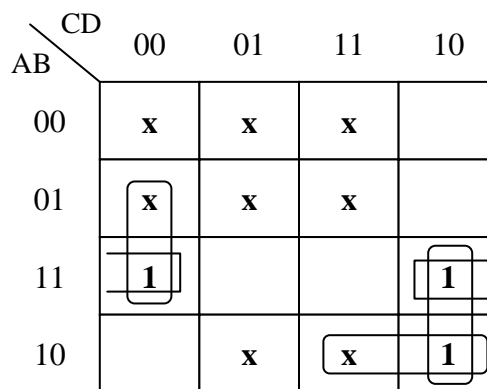


Figure 3.5: K-map for example 1.

Table 3.2: Selected cubes covering for example 1.

Minterm	Minterm Probability	-100	11-0	1-10
1010	0.23			1
1100	0.045	1	1	
1110	0.68		1	1

Although the two covers contain two cubes, there is a difference in reliability. Considering the two cubes i.e., -100 and 1-10, each minterm is covered by only one cube. And so, for each minterm, if an error hits in the logic of any cube it will not be masked,

and it will be propagated to the output. In the second implementation i.e., 11-0 and 1-10, the minterms 1010 and 1100 are also covered by one cube, but the minterm 1110 is covered by the two cubes. That is, for the 1110 minterm, if an error hits in the logic of one cube, it will be masked by the other cube.

Figure 3.6 (a, b) shows the logical circuit for implementations 1 and 2. In implementation 1, each minterm is covered by only one cube, and so, if an error hits and changes the output of the covering cube, this error will not be masked. The 1010 minterm is only covered by the cube (1-10). The number of faulty wires for this minterm is 8. The 1100 minterm is only covered by the cube (-100) with 8 faulty wires. The 1110 minterm is also covered by one cube (1-10) and has 7 faulty wires. In the second implementation, the number of faulty wires for minterms 1010 and 1100 are 8. Consider the minterm 1110 in implementation 2, since this minterm is covered by the two cubes, if a single error hits in any cube, it will be masked by the other cube, except for fan-outs that are feeding both cubes. Figure 3.7 shows the faulty wires for the 1110 minterm in implementation 2, they are 4 wires, one of them is due to output wire which can't be masked in all cases, the second is due to fan-out of the first input, and the remaining two are due to fan-out of the fourth input, which is an INVERTER. The faulty wires for minterm 1110 are reduced from 7 to 4. Considering the large probability of 1110 minterm, this reduction has a high impact on reducing the failure rate of the circuit.

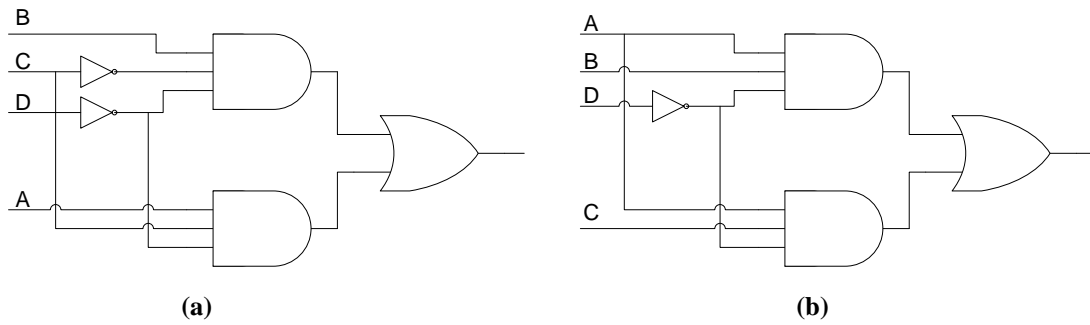


Figure 3.6: Different implementations for example 1.

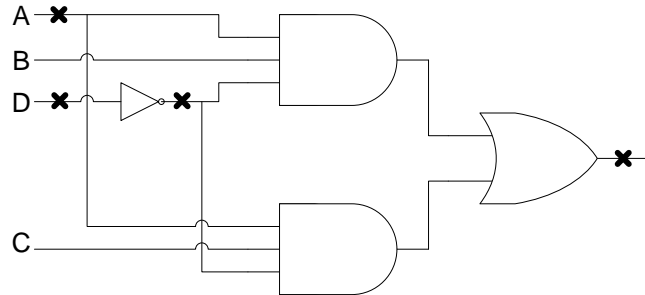


Figure 3.7: Faulty wires in example 1 for minterm 1110 for second implementation.

To compare between the two different implementations, the failure rate for each one of them can be estimated using equation (1). By ignoring failure rate due to off-set minterms, the failure rate due to on-set minterms for the two implementations can be calculated as follows:

$$\text{Failure Rate 1} = \frac{[(0.23 * 8 + 0.045 * 8 + 0.68 * 7)]}{13 * 2} = 0.268$$

$$\text{Failure Rate 2} = \frac{[(0.23 * 8 + 0.045 * 8 + 0.68 * 4)]}{12 * 2} = 0.205$$

According to estimated failure rates, the second implementation reduces the failure rate dramatically. This is due to the extra covering for the minterm 1110 which has a very high probability of occurrence.

To find the covering cubes, the essential cubes are found first and added to the cover. An essential cube is defined as a cube that covers at least one minterm that is not covered by any other prime cube. Essential cubes are mandatory, and they are added to the cover first. After that, cubes are added according to their weight. At each time, cubes' weights are calculated and the cube with the maximum weight is added to the cover if it covers new uncovered minterms. This process terminates when all minterms are covered. If there are no essential cubes, the cube that covers more minterms (considering their probabilities) is added first. After covering all minterms, the redundant cubes are removed from the cover, since at this stage we want a cover that has no redundant cubes. Redundant cubes can be added in the next stage explained in the next sub-section. This is important since not all redundant cubes have the same effect on error masking.

Since we want -by adding a new cube- to decrease the number of faulty wires considering probabilities of minterms, the weights for cubes will be calculated as follows. Each not selected cube will be assigned a weight that represents the enhancement on fault masking, i.e., decreasing of faulty wires. For each minterm, a weight is calculated due to each cube, then for each cube, the weights of all minterms are summed. For a cover  $S$ , minterm  $M$ , and cube  $C$ ,  $F_{old}$  represents the number of faulty wires in the cover  $S$  for  $M$ , and  $F_{new}$  represents the number of faulty wires in cover  $S$  with the new cube  $C$  added to the cover. The weight of the minterm  $M$  can be calculated as:

$$W_M = \left\{ \begin{array}{ll} 0.00001 & \text{if } M \text{ is covered by } C \text{ and is not covered by } S \\ \frac{F_{old} - F_{new}}{F_{old}} & \text{if } M \text{ is covered by } S \end{array} \right\} \quad (2)$$

All not selected cubes will be assigned a weight using equation (2). If the minterm  $M$  is covered by cube  $C$ , and is not covered by any cube in the cover  $S$ , a very small value is used for weight. This condition is needed when the function has no essential implicants. In such case, the first cube to be selected will be the cube with the highest probability of occurrence. The value is very small so that it will not affect selecting a cube according to covering, and selection will be based on higher probability of occurrence. However, if the minterm  $M$  is covered by at least one cube in the cover  $S$ , the weight is assigned the value  $(F_{old} - F_{new})/F_{old}$ . This value represents the masking improvement that cube  $C$  will add regarding minterm  $M$ .

Calculating  $F_{old}$  and  $F_{new}$  is done using the same way used in the two-level estimator when finding the number of faulty wires for on-minterms. According to the value of  $W_M$  for each minterm, and by assuming that the probability of a minterm to occur is  $M_p$ , the weight of a cube is as follows:

$$W_C = \sum_{M \in \text{on-minterms}} W_M * M_p \quad (3)$$

Consider the k-map for a circuit in Figure 3.8. The covering matrix for all prime implicants is built first as shown in Table 3.3. Initially, the cover set is empty, i.e.,  $Cover = [ ]$ . The essential cube (011-) will be added first to the cover,  $Cover = [ 011- ]$ . Then, the weight for each of the remaining cubes is calculated. The cube (-111) will have the highest weight, since for now only two minterms are covered (0111 and 0110). The cube

(-111) covers one of them, and so it will decrease the number of faulty wires. Hence,  $Cover = [011-, -111]$ . The next selected cube is (1--1) and  $Cover = [011-, -111, 1--1]$ . At this stage, all minterms are covered. By checking redundancy, it's clear that the cube (-111) is redundant. The final cover will be:  $Cover = [011-, 1--1]$ .

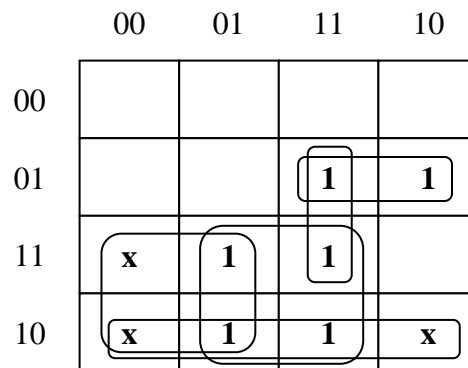


Figure 3.8: K-map for example 2.

Table 3.3: Cubes covering for example 2.

Minterm	Minterm Probability	011-	-111	1-0-	1--1	10--
0111	0.1	1	1			
0110	0.1	1				
1101	0.4			1	1	
1111	0.2		1		1	
1001	0.1			1	1	1
1011	0.01				1	1



### 3.3.4 Adding Redundant Cubes

After finding an irredundant set of cubes that cover all minterms, a set of redundant cubes can be added to the cover to increase fault masking. Adding redundant cubes is done according to maximum weight also. A specified number of redundant cubes can be added to the cover, or a weight threshold can be specified for adding a redundant cube. In this work, a threshold is specified, such that the algorithm keeps adding redundant cubes as long as the added cube satisfies the threshold. The threshold represents the total failure rate improvement for the sub-circuit resulting from adding a redundant cube. The weight for each cube is calculated as explained in the previous sub-section.

Suppose that the cube with the maximum weight is  $C_{MAX}$ ,  $S$  is the covering set before adding  $C_{MAX}$ , and  $S_I$  is the cover after  $C_{MAX}$  is added to the cover,  $F_S$  is the estimated failure rate of  $S$ , and  $F_{S_I}$  is the estimated failure rate of  $S_I$ . Then, the cube is added to the cover as a redundant cube if it satisfies the following:  $(F_S - F_{S_I} > Thr)$ . We have used 0.01 as a suggested value for  $Thr$ . The threshold value represents how much a redundant cube will decrease the failure rate of a circuit.  $F_S$  and  $F_{S_I}$  are computed using the two-level single fault failure rate estimator given in Figure 3.4. Following on the previous example, three cubes can be used as redundant cubes (-111, 1-0-, and 10--). If the cube 1-0- is added as a redundant cube, the minterms 1101 and 1001 will be covered by 2 cubes. Those minterms have high probability of 0.5. If the cube -111 is used, minterms of total probability of 0.3 will be covered by 2 cubes. Using the cube 10--, minterms of total probability of 0.11 will be covered by 2 cubes. Hence, the redundant cubes sorted according to their weights are: 1-0-, -111 and 10--. It's worth to mention that cube -111 was selected as the highest weight cube after the essential in the previous phase (finding

cover). Now, another cube has the maximum weight. This emphasizes the need of deleting redundant cubes in the previous phase, and then adding extra redundant cubes according to their weight.

### 3.3.5 Duplicate One Cube Phase

For some extracted sub-circuits, the phase with the higher probability of occurrence produces only one cube. This case happens when a phase produces only one prime implicant. In such case, this cube will not help in reliability. Consider a circuit with a very high off-set probability with the implementation shown in Figure 3.9(a). Most of the time the NAND will produce logic 0; if any error hits at the inputs, there is no way for a fault to be masked, and it will generate a faulty output. To increase the reliability in such case, this single cube can be duplicated. Figure 3.9 (b) shows the duplication scheme, in which two copies of the cube are used to feed an AND gate. The fault masking in the duplication scheme will be as follows. Consider first the off-set minterms which have the higher probability. If an error hits at any NAND gate logic either inputs or output, it will be masked by the other NAND gate, since we have an AND gate at output. Regarding the opposite phase, on-set phase, masking only occurs at the inputs of NAND gates.

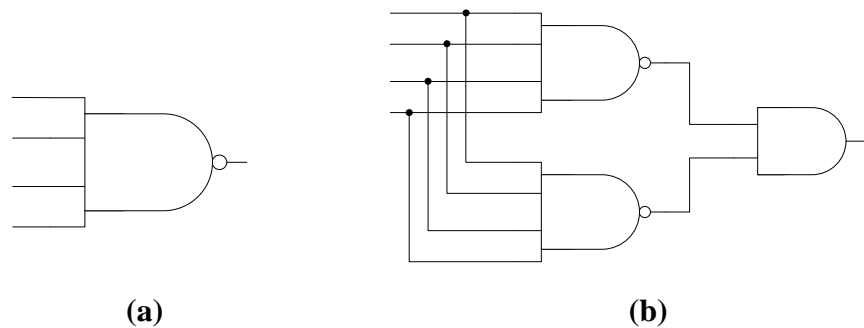


Figure 3.9: Duplication of one cube off-phase.

However, if the higher probability phase is the on-set, and it produces only one cube, i.e., one AND gate, then the logic is duplicated using an OR gate. Figure 3.10 (a) shows an implementation of on-phase with one cube only, and the duplication scheme is shown in Figure 3.10(b).

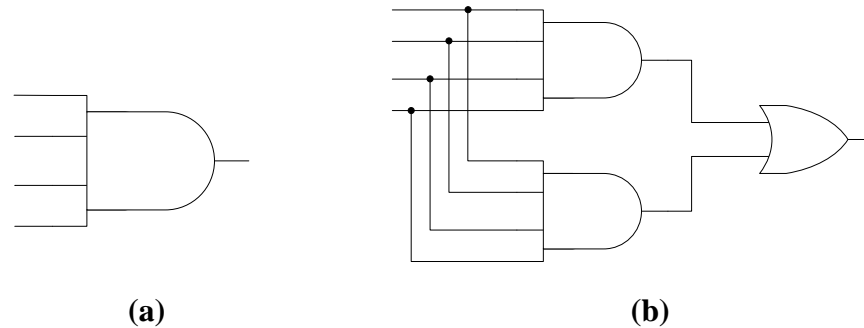


Figure 3.10: Duplication of one cube on-phase.

### 3.3.6 Algorithm 1: Two-Level Circuit Synthesis

Each extracted sub-circuit will be synthesized using Algorithm 1 which is shown in Figure 3.11. Algorithm 1 represents a two-level synthesis technique that uses a heuristic to increase fault masking for high probability input vectors for an extracted circuit.

In Figure 3.11, inputs are: the minterms of ON and OFF phases with their probabilities, don't care minterms, and the threshold for adding redundant cubes. First, the algorithm will determine the phase to implement. This is done by computing the probability of each phase, and selecting the phase with higher probability of occurrence. Next, all prime implicants according to minterms of selected phase are computed using Espresso tool. The essential primes are computed and added to the cover  $S$ .

---

**inputs:** on-set and off-set minterms with their probabilities, don't cares, threshold  $Thr$ .

1.  $P_{ON}$  = summation of probabilities of all *On-set minterms*.  
 $P_{OFF}$  = summation of probabilities of all *Off-set minterms*.
2. **if** ( $P_{ON} > P_{OFF}$ )  $SM = On\text{-set minterms}$  **else**  $SM = Off\text{-set minterms}$ .
3. Initialize the cover  $S = \{ \}$ .
4. Using  $SM$ :
  - 4.1. Find all prime implicants (cubes)  $P$  using Espresso tool.
  - 4.2. Find all essential cubes (if exist) and add them to the cover  $S$ .
5. **while** (not all minterms in  $SM$  are covered by  $S$ ) {
  - Calculate weight for each not selected prime implicant in  $P$  using eqn. 3 using  $SM$  minterms.
  - Select the highest weight cube  $C$  that covers new minterms in  $SM$  that are not covered by  $S$ .
  - Add cube  $C$  to the cover  $S$ .
 } // end while
6. For every cube  $c_i$  in the cover  $S$ : // remove redundant cubes
  - if** (each minterm covered by  $c_i$  is covered by at least one other cube in  $S$ )  
 delete cube  $c_i$  from the cover  $S$ .
7. **while** (true) { // add redundant cubes
  - Calculate weight for each not selected prime implicant in  $P$  using eqn. 3.
  - $R$  = the prime cube with highest weight.
  - $F_S$  = estimated failure rate of the cover  $S$  using eqn.1.
  - $F_{S+R}$  = estimated failure rate of the cover  $S + R$  using eqn.1.
  - if** ( $F_S - F_{S+R} > Thr$ ) add the  $R$  cube to the cover  $S$  **else** Go to step 8.
 } // end while
8. Initialize synthesized circuit  $new$ .
9. **if** ( $|S| = 1$ ) // if the cover  $S$  produces only one cube; duplicate.
  - if** ( $P_{ON} > P_{OFF}$ )  $new$  = cube (AND) gate is duplicated using an OR gate.
  - else**  $new$  = cube (NAND) gate is duplicated using an AND gate.**else**{ // the cover  $S$  contains multiple cubes
  - if** ( $P_{ON} > P_{OFF}$ )  $new$  = implement  $S$  cubes as AND gates feeding an OR gate.
  - else**  $new$  = implement  $S$  cubes as AND gates feeding a NOR gate.
 }
10. Return ( $new$ ).

---

Figure 3.11: Algorithm 1- Two-level tool to enhance fault tolerance.

Then, cubes are added to the cover according to a weight metric. A weight for each cube not included in the cover is calculated, and the cube with the highest weight is selected and added to the cover if it covers newly uncovered minterms, otherwise, the next highest weighted cube is selected and so on. This process of adding cubes is terminated when all minterms (of the selected phase) are covered. After that, the redundant cubes are removed from cover. In the next step, a set of cubes are added as redundant cubes to increase fault masking. Redundant cubes are added according to their weights; adding redundant cubes process is terminated when the decreasing of failure rate for a circuit resulting from a redundant cube is less than the redundancy threshold. Finally, the implementation of the new circuit is formulated according to the type of selected phase. Duplication is applied when the cover produces only one cube.

### 3.3.7 Complexity of Algorithm 1

The complexity of Algorithm 1 is mainly controlled by the number of inputs of an extracted circuit. Complexity can be explored as follows:

- Number of prime implicants has an upper bound of  $O(3^k/k)$  [49], where  $k$  is the number of circuit inputs.
- Computing the weight of a cube needs traversing all minterms; it has a complexity of  $O(M)$ , where  $M$  is the number of minterms, which is  $O(2^k)$ .
- The time consuming step in Algorithm 1 is finding the cover:
  - Covering all minterms step has two nested loops. The outer loop terminates when all minterms are covered; it has a complexity of  $O(C)$ ,

where  $C$  is the number of covering cubes. The inner loop goes through all generated prime cubes and computes a weight for each cube; it has a complexity of  $O(3^k/k * M)$ . The complexity of covering all minterms is  $O(3^k/k * M * C)$ .

- The complexity of removing redundant cubes is  $O(C^2 * M)$ , where  $C$  is the number of covering cubes.
- The complexity of adding redundant cubes is  $(R * 3^k/k * M)$ , where  $R$  is the number of extra redundant cubes

Usually the number of extra redundant cubes is small ( $R \ll C$ ), the complexity of finding cover is  $O(3^k/k * M * C)$ . The upper bound of  $C$  is  $O(3^k/k)$ . Therefore, the upper bound of the complexity of Algorithm 1 is  $O(3^k/k * 2^k * 3^k/k) = O(18^k/(k^2))$ . As the value of  $k$  is 15 at maximum, this time complexity is feasible.

### 3.4 Conclusion

In this chapter, we have introduced a novel technique, presented in Algorithm 1, to increase combinational circuit reliability and hence fault tolerance. This technique relies on enhancing fault masking for each extracted sub-circuit, and hence the overall fault masking of the original circuit is improved as well. We introduce a heuristic to find a two-level cover that maximizes fault tolerance especially for input patterns with high probability of occurrence for each extracted sub-circuit.

We found that the selection of phase to implement is very important, since selecting a phase with very low probability to occur will have negligible improvement. Most of extracted sub-circuits from our experiments either have a very high on-phase probability or a very high off-phase probability. In such cases, selecting the phase with the higher probability is effective.

According to our knowledge, none has presented the introduction of finding covers for extracted circuits that maximizes fault masking.

## Chapter 4

# **MULTI-LEVEL LOGIC SYNTHESIS FOR SOFT ERROR TOLERANCE**

In this chapter, we will investigate the effect of multi-logic synthesis transformations on soft error masking. We will restrict ourselves to the fast extraction algorithm [50]. Fast extraction is a very efficient extraction method; it's based on the extraction of double-cube divisors along with their complements and single cube divisors with two literals. To minimize area, single cube and double cube divisors with highest area weights are selected. In this chapter, we will use fast extraction to minimize area of circuits' obtained in Chapter 3, but by avoiding extracting cubes that have a negative impact on reliability.



## 4.1 Fast Extraction for Area Optimization

Fast extraction transformation involves extracting two types of cubes: double-cube divisors and single-cube divisors. Double-cube divisors are cube-free multiple cube divisors having exactly two cubes. Suppose we have the following Boolean function written as sum-of-products:

$$f = ade + ag + bcde + bcg$$

Table 4.1 shows all possible double-cube divisors that can be extracted from this function along with their bases. For instance, if the first double cube is to be extracted, the multi-level network of the function  $f$  will be as follows:

$$t = de + g$$

$$f = ta + tbc$$

If number of literals is used as an approximation of area cost, then a saving of 4 literals (12 - 8) is obtained.

Table 4.1: Double-cube divisors along with their bases for  $f = ade + ag + bcde + bcg$ .

Double-cube divisors	Base
$de + g$	$a, bc$
$a + bc$	$g, de$
$ade + bcg$	$\{-\}$
$ag + bcde$	$\{\}$

Single-cube divisors of interest are those with two literals only; for previous function  $f$ , two single cube divisors can be extracted:  $bc$  and  $de$ . If  $bc$  is extracted then the multi-level network of the function  $f$  will be as follows:

$$t = b c$$

$$f = a d e + a g + t d e + t g$$

Area saving in this case will be zero. Single cubes with two literals can have a positive area weight if it's included in more than two cubes.

Figure 4.1 shows the fast extraction algorithm used to get multi-level networks with optimized area. It extracts all possible double and single cube divisors and calculates the area weight for each cube. The cube with the highest area weight will be extracted. Then, the same algorithm will be applied recursively on the resulting multi-level network until no positive area weight cubes can be extracted. The area weight for a single cube divisor  $s$  can be simply calculated according to the following equation:

$$W_s = k - 2$$

Where  $k$  is the number of cubes containing the single cube  $s$ . However, the area weight of a double-cube divisor  $d$  is calculated using the following formula:

$$W_d = p * l - p - l + B + c$$

Where  $p$  is the number of bases including complement bases,  $l$  is the number of literals in cube  $d$ ,  $B$  is the total number of literals in all bases, and  $c$  is the number of cubes containing the complement of double-cube divisors with two literals.

**Repeat**

Select a double-cube divisor  $d$  that has a maximum area weight  $W_{dmax}$ .

Select a single-cube divisor  $s$  having a maximum area weight  $W_{smax}$ .

**If**  $W_{dmax} > W_{smax}$

Select  $d$ .

**Else**

Select  $s$ .

$W = \max (W_{dmax}, W_{smax})$ .

If  $W > 0$  then substitute selected divisor.

Re-compute weights of affected single-cube and double-cube divisors.

**Until** ( $W \leq 0$ )

---

Figure 4.1: Fast Extraction algorithm for area optimization [50].

## 4.2 Fast Extraction for Reliability

In this section, we will investigate using fast extraction for reliability. The objective here is to obtain area improvement but with a low impact on reliability. The effect on fault masking of both single and double cubes will be studied.

To study the effect of single-cube extraction on masking, consider the Boolean function  $f1 = abc + abd + abe + kg$ . The single-cube that can be extracted is  $ab$ , with area weight of 1. The multi-level network after extracting  $ab$ , will be as follows:

$$t = ab$$

$$f1 = tc + td + te + kg$$

Figure 4.2 (a) shows the logic network for the original function  $f1$ , and Figure 4.2 (b) shows the logic network for  $f1$  after extracting the single-cube  $ab$ . Considering the case for on-set minterms covered by more than one cube, the masking won't be affected in the

cubes from which the single-cube was extracted. That is, if a fault hits in the logic of one cube, it's still masked by other cubes that cover the same input minterm. For on-minterms covered by only one cube, suppose that  $S$  is the set of cubes from which the single-cube divisor is extracted. Two cases can be analyzed: The first one, if the covering single cube is among cubes in  $S$ . The other case, if the single cube is not contained in  $S$ . In the first case, the circuit behavior against single faults won't be affected. The faults in other cubes are masked by the covering cube, and the faults in the covering cube are not tolerated. In the second case, extracting the single-cube divisor won't affect masking behavior as the covering cube is not included in  $S$ , and so, faults in the logic of other cubes will be masked by the output of the covering cube.

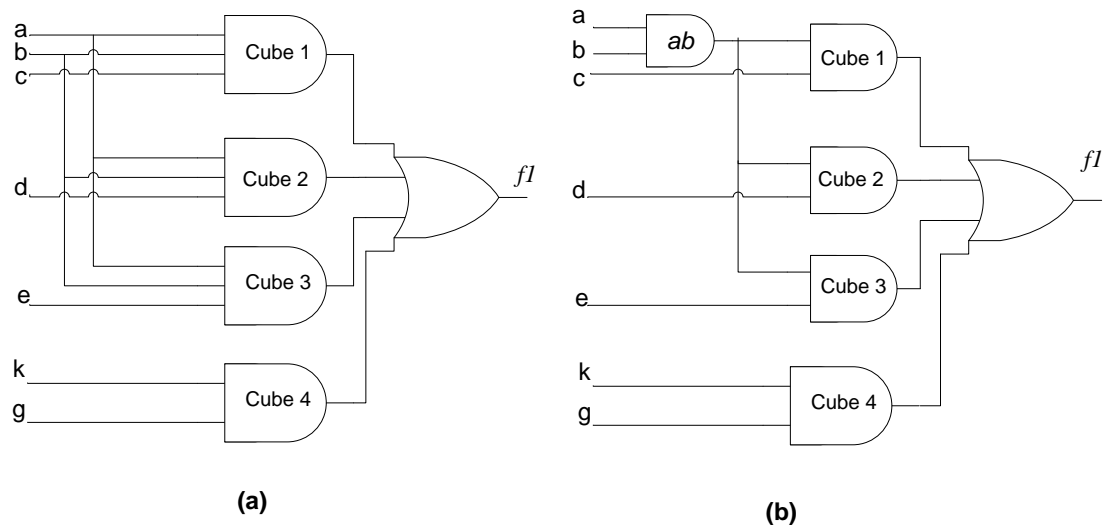


Figure 4.2: Single-cube extraction example: (a)  $fl = abc + abd + abe + kg$ , (b)  $t = ab$ ,  $fl = tc + td + te + kg$ .

To analyze the effect of extracting a single-cube divisor on off-minterms, consider the off-minterm  $abcdekg = 0011100$ . In Figure 4.2 (a), if this minterm is applied, then any single fault hitting at the inputs of the AND gates is tolerated since each cube has two

zero inputs. After extracting the single-cube divisor as shown in Figure 4.2 (b), each cube from 1 to 3 has two inputs, and only one of them is zero. In this case, single faults at zero inputs of these cubes are not tolerated. Thus, the circuit becomes worse in term of reliability. And so, the off-minterms in which circuit reliability maybe affected are those minterms that make both extracted literals to be zeros. In each cube from which the single-cube divisor is extracted, masking at the inputs is affected if only two inputs are zeros, and those inputs are the ones extracted in the single-cube divisor. Based on previous discussion, for a single-cube divisor  $s$ , a reliability weight can be defined as follows:

$$R_s = \sum_{m=1}^n \left( P_m * \frac{|C_m|}{|C|} \right)$$

$n$  is the number of off-minterms that make both extracted literals of single cube to be zeros.  $P_m$  is the probability of off-minterm  $m$ .  $C$  is the set of cubes from which the single-cube divisor is extracted.  $C_m$  is the set of cubes from  $C$  such that if minterm  $m$  is applied, the cube will have only two zero inputs, and those inputs are the ones extracted in the single-cube divisor.

Extracting double-cube divisors has a different impact on fault masking. Consider the Boolean function  $f_2 = ab + ac + de$ , with the double-cube divisor  $(b + c)$ . The multi-level network after extracting  $(b + c)$  will be as follows:

$$t = b + c$$

$$f_2 = a t + d e$$

Figure 4.3 (a) shows the logic network for the original function  $f_2$ , and Figure 4.3 (b) shows the logic network for  $f_2$  after extracting the double cube  $b + c$ . Considering the case for on-set minterms covered by more than one cube, fault masking according to covering is different than that in the original implementation. To illustrate the difference, suppose  $m_1$  is a minterm that is covered only by  $\text{cube}_1$  and  $\text{cube}_2$ . In the original implementation, if an error hits in the logic of any cube, it will be masked by the other cube except if the error hits in the shared input  $a$ . However, in the second implementation, masking of errors according to covering can only occur at the inputs of OR gate ( $b + c$ ). It's clear that none of the errors at inputs or output of AND1 are tolerated. Note that the error on input  $a$  will not be masked in both implementations. And so, in this case, we are adding two extra faulty lines, namely, the output of AND1 and its second input. In another case, suppose  $m_2$  is a minterm that is covered by  $\text{cube}_3$  and at least one of the cubes;  $\text{cube}_1$  and  $\text{cube}_2$ . In this case, fault masking isn't affected since  $m_2$  is covered also by another cube outside the extracted double cube base cubes, and so, if an error hits in any of the AND gates; AND1 and AND2 or their inputs, it will be masked by the other AND.

To analyze the affect on on-minterms covered by only one cube, suppose that  $S$  is the set of cubes from which the double-cube divisor is extracted. If the single cube isn't among cubes  $S$ , masking behavior is not affected after extracting the double-cube divisor. The faults in other cubes are masked by the covering cube, and the faults in the covering cube are not tolerated. If the single cube is contained in  $S$ , masking behavior is also not affected. The faults in the covering cube logic are not tolerated, and the faults in other

cubes logic are masked by the covering cube. Regarding off-minterms, masking properties for extracted cubes if they have multiple zero inputs are still preserved.

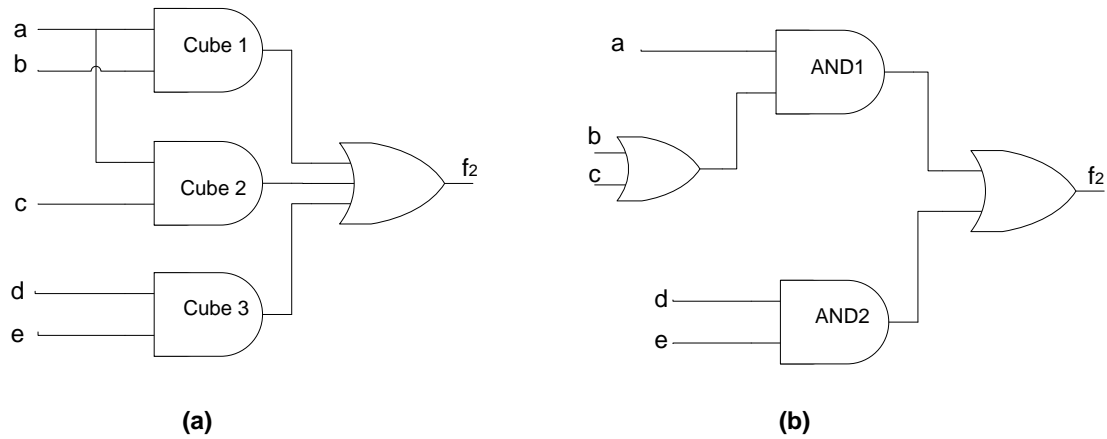


Figure 4.3: Double-cube extraction example: (a)  $f_2 = ab + ac + de$ , (b)  $t = b+c$ ,  $f_2 = at + de$ .

To test the previous analysis, both implementations were simulated using minterms with probabilities as specified in Table 4.2. In this table, four on-minterms are provided with their probabilities. We assume that the probability of other on-minterms is 0.1, and the probability of off-minterms is 0.1. This table shows that the probability of minterm 11100 has the main effect on failure rate of the multi-level implementation. As the probability of this minterm increases, the failure rate of the multi-level implementation becomes worse. This can be generalized as follows: failure rate of a multi-level circuit resulting from extracting a double-cube depends on the probability of minterms that are covered only by cubes from which the double-cube is extracted.

Table 4.2: Failure rate for two implementations of  $f_2 = ab + ac + de$  for different minterms probabilities.

Minterms	Covered By	Minterms' Probabilities						
		0.77	0.01	0.01	0.25	0.15	0.05	0.01
11000	cube1 only	0.77	0.01	0.01	0.25	0.15	0.05	0.01
11100	cube1, cube2	0.01	0.01	0.01	0.05	0.35	0.65	0.77
11011	cube1, cube3	0.01	0.77	0.01	0.25	0.15	0.05	0.01
00011	cube3 only	0.01	0.01	0.77	0.25	0.15	0.05	0.01
Failure Rate	Original	0.23	0.05	0.18	0.14	0.13	0.11	0.09
	After extracting double cube	0.25	0.06	0.22	0.18	0.22	0.2	0.22

To analyze the case which involves extracting a double-cube along with its complement, three cases can be considered:  $D_{112}$ ,  $D_{222}$ ,  $D_{223}$ . An example for the  $D_{112}$  case is a double-cube  $(a+b)$  with its complement  $a'b'$ . An example of the  $D_{222}$  case is a double-cube  $(ab + a'b')$  with its complement  $(ab' + a'b)$ . An example of the  $D_{223}$  case is a double-cube  $(ab + a'c)$  with its complement  $(ab' + a'c')$ . For the cases  $D_{222}$  and  $D_{223}$ , it's impossible for a minterm to be covered by the two cubes from which a double-cube divisor is extracted neither its complement cubes. Hence, fault masking will not be affected if the double-cube divisor is extracted in these two cases. For the  $D_{112}$  case, consider the Boolean function  $f_3 = ac + bc + abd$ . The double-cube divisor that can be extracted is  $(a + b)$ . The multi-level network after extracting  $(a + b)$  along with its complement  $(a'b')$  will be as follows:  $t = a + b$ ,  $f_3 = tc + t'd$ . Figure 4.4 (a) and (b) show the logic network for  $f_3$  before and after extracting the double-cube respectively. As discussed before, if a minterm is covered by both cubes  $ac$  and  $bc$  (cube 1 and cube2), fault masking for this minterm will be highly affected by extraction.



For minterms covered by only one cube, two cases can be considered: a case for a minterm covered by one of the cubes cube1 or cube2, and a case of a minterm covered by cube3. In both cases, masking behavior is not changed since only AND gate will produce logic 1. However, the number of faulty wires may slightly increase in the multi-level network depending on the value of the minterm. Nevertheless, minterms covered by both cubes (cube1 and cube2) are the main minterms for which masking behavior is changed. Regarding off-minterms, the same analysis discussed before can be applied here.

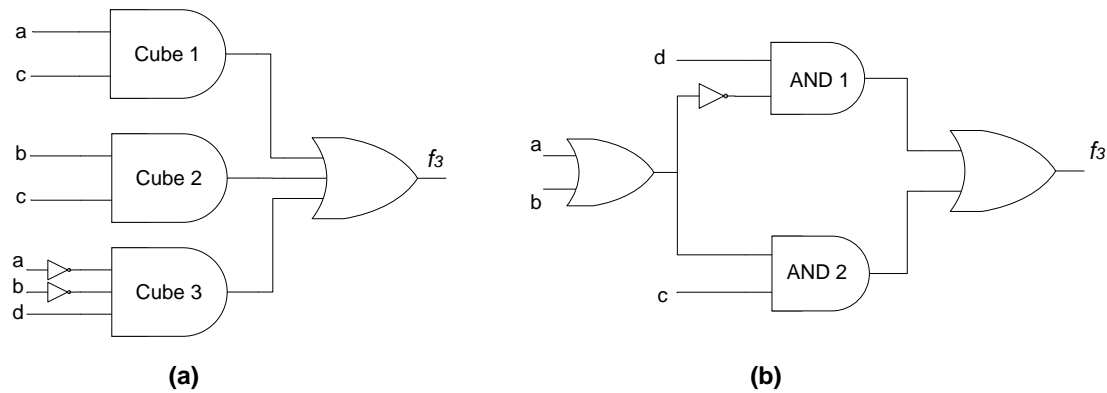


Figure 4.4: Double-cube extraction with complement example: (a)  $f_3 = ac + bc + abd$ , (b)  $t = a + b$ ,  $f_3 = tc + t'd$ .

As the objective is to improve area cost with low impact on reliability, double-cube divisors should be extracted such that this objective is satisfied. Each double-cube divisor can have a reliability weight in addition to the area weight. For a double-cube  $d$  with bases  $b_1, \dots, b_n$ , the reliability weight can be calculated as follows:

$$R_d = \sum_{i=1}^n R_{b_i}$$

$R_{b_i}$  is the probability of minterms covered only by cubes from which base  $b_i$  is extracted. According to this weight, a double-cube divisor will not be extracted if its reliability weight is larger than a certain threshold.

To justify the proposed weight, consider the following Boolean function:  $x = abf + adef + cde$ . Two double cubes can be extracted:  $(b + de)$  and  $(af + c)$ . If  $(b + de)$  is extracted then the multi-level circuit will be as follows:  $t = b + de$ ,  $x_1 = taf + cde$ . If  $(af + c)$  is selected then the multi-level circuit will be as follows:  $t = af + c$ ,  $x_2 = abf + tde$ . Table 4.3 shows the failure rate for the  $x$ ,  $x_1$ , and  $x_2$  implementations for different minterms' probabilities. In this table, two on-minterms are provided with their probabilities. We assume that the probability of other on-minterms is 0.1, and the probability of off-minterms is 0.1. According to suggested reliability weight, the weight of  $(b + de)$  double cube is the probability of the minterm 110111, and the weight of  $(af + c)$  is the probability of 101111. The table shows that extraction of the double-cube with lower weight gives better circuit. And so, the proposed reliability weight chooses the cube with better reliability. The weight function reflects the negative impact on reliability resulting from extracting the double-cube divisor.

Table 4.3: Failure rate for two implementations of  $x = abf + adef + cde$  for different minterms probabilities.

Minterm	Covered by	Minterms Probabilities				
		0.1	0.3	0.4	0.5	0.7
110111	$abf$ and $adef$	0.1	0.3	0.4	0.5	0.7
101111	$adef$ and $cde$	0.7	0.5	0.4	0.3	0.1
Failure Rate	$x$	0.084	0.094	0.081	0.09	0.083
	$x_1$	0.11	0.122	0.145	0.17	0.178
	$x_2$	0.18	0.168	0.142	0.124	0.1

### 4.3 Algorithm 2: Enhancing Area for Reliability Optimized Two-Level Circuits

Given a reliability optimized two-level circuit, Algorithm 2 - shown in Figure 4.5- can be used to find a multi-level implementation with less area overhead and with low impact on reliability. Algorithm 2 is a modified version of the original fast extraction algorithm. Instead of extracting the single-cubes and double-cubes according to area weight only, the reliability weight is taken into consideration.

Algorithm 2 finds the set of all possible double-cube divisors; the area and reliability weights are calculated for each of them. A set of double-cube divisors  $D$  with reliability weight less than a threshold is constructed. Among double-cube divisors in  $D$ , the best double cube  $d$  to be extracted is the one with highest area weight. For all possible single-cube divisors, the set of single-cube divisors  $S$  with reliability weight less than a threshold is constructed. Among single-cube divisors in  $S$ , the best single-cube  $s$  to be extracted is the one with highest area weight. The cube among  $s$  and  $d$  with larger area saving is selected to be extracted. The same process is repeated until no area improvement can be achieved.

---

**inputs:** two-level circuit, Threshold.

**Repeat**

Find all double-cube divisors  $D$  with Reliability Weight  $<$  Threshold.

Select a double-cube divisor  $d$  from  $D$  that has a maximum Area weight  $W_{dmax}$ .

Find all single-cube divisors  $S$  with Reliability Weight  $<$  Threshold.

Select a single-cube divisor  $s$  having a maximum Area weight  $W_{smax}$ .

**If**  $W_{dmax} > W_{smax}$

    Select  $d$ .

**Else**

    Select  $s$ .

$W = \max (W_{dmax}, W_{smax})$ .

    If  $W > 0$  then substitute selected divisor.

    Re-compute weights of affected single and double cube divisors.

**Until** ( $W \leq 0$ )

---

Figure 4.5: Algorithm 2.

## 4.4 Framework of Enhancing Fault Tolerance Using Algorithms 1 and 2

Algorithm 1 discussed in Chapter 3 is used to enhance fault masking for an extracted sub-circuit. It results in a two-level circuit in which masking against single fault is maximized. Algorithm 2 improves area overhead of reliability optimized two-level circuits. It has low impact on reliability enhancement achieved using Algorithm 1. The following framework should be applied in the design procedure, as shown in Figure 4.6:

1. Given a combinational circuit represented in a multi-level network of gates.
2. Extract sub-circuits.
3. Find CDCs and probabilities of care minterms for the extracted circuits.
4. Re-synthesize each extracted circuit using Algorithm 1.

5. Apply Algorithm 2 on each two-level synthesized circuit to enhance area overhead.
6. Merge the re-synthesized circuits into the original circuit.

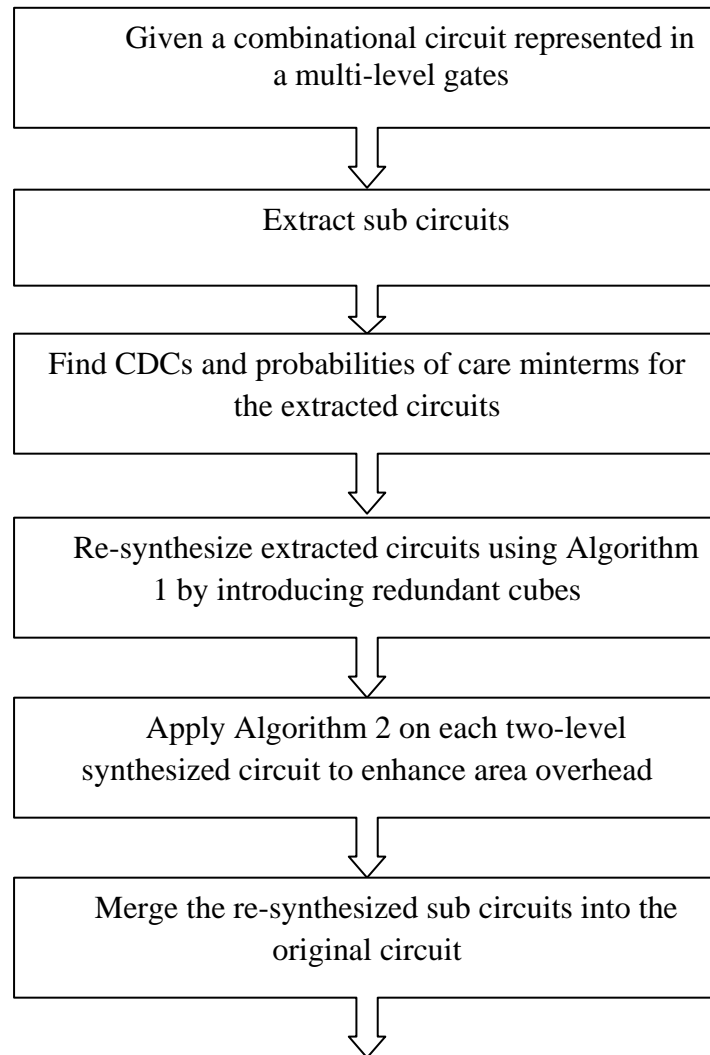


Figure 4.6: Framework of Algorithm 1 and Algorithm 2.

## 4.5 Conclusion

In this chapter, fast extraction algorithm is explored to enhance area for reliability optimized two-level circuits. It's found that this algorithm with some modifications can be used effectively to achieve this objective. In addition to area weight, reliability weight is also used in selecting single-cube and double-cube divisors. This technique is proposed as Algorithm 2. Algorithm 2 ensures that the selected single-cube and double-cube divisors will have a low negative impact on reliability.

## Chapter 5

# **SIMULATION ENVIRONMENT AND FRAMEWORK**

To show the impact of our proposed algorithms in Chapters 3 and 4, a reliability evaluator based on Monte-Carlo simulation is implemented. In this chapter, we will describe the simulation environment that our reliability evaluator uses to report fault tolerance measurement of benchmark combinational circuits. In the following sections, we will describe how to measure combinational circuit reliability. Some assumptions regarding fault tolerance evaluator has been made and listed too. After that, a stuck-at fault model is chosen and the fault injection mechanism is described.

## 5.1 Measuring Combinational Circuit Reliability

Comparing the reliability of original combinational benchmark circuits with their re-synthesized versions, using our methods, or with other techniques is done by measuring the failure rate per injected faults. Knowing that reliability is inversely proportional to failure rate, varying the number of faults injected to the combinational benchmark circuits and measuring the failure rate for each injected fault is an effective way to measure their fault tolerance (or reliability). Logically, circuits' reliability starts from 100% when no faults occur and decrease as the number of faults increases. In other words, the failure rate grows as the number of faults increases.

To evaluate circuit failure rate probability, a simulation-based reliability model as the one used in [51] is often adopted. However, since we eventually plot the failure rate against the number of injected faults, we modified it according to our need as it will be explained in the next section.

## 5.2 The Simulation Framework of Reliability Evaluator

The goal of our implemented reliability evaluator is to measure the failure rate of combinational circuits as the number of injected faults grows. It is done by using Monte-Carlo simulation methods. For each circuit, we find the failure rate by injecting faults for a certain number of iterations and counting the number of success and failed iterations. The framework of getting the failure rate of a circuit is stated in the following points:



- For every number of faults  $F$  from 1 to  $N$ , where  $N$  is the maximum number of faults to be injected:
  - Set the number of failed iterations,  $k$ , to zero.
  - For each iteration  $i$  from 1 to  $T$ , where  $T$  is the maximum number of iterations:
    - Generate a random input vector, where the length of the vector is the same as the number of inputs in the circuit.
    - Simulate the circuit to get fault free original output by applying the generated random vector and store the output response in  $O$ , where the length of  $O$  is the same as the number of outputs in the circuit.
    - Inject  $F$  number of faults in the circuit randomly.
    - Simulate the circuit to get the faulty output response and store it in  $O_F$ , where the length of  $O_F$  is the same as the number of outputs in the circuit.
    - If the output response  $O$  is different from  $O_F$ , then  $k$  is incremented. Otherwise it is successful.
  - Repeat for next  $i$ .
  - Calculate the failure rate when  $F$  faults are injected, denoted by  $R_F$ , by
 
$$R_F = \frac{k}{T}.$$
- Repeat for next  $F$ .

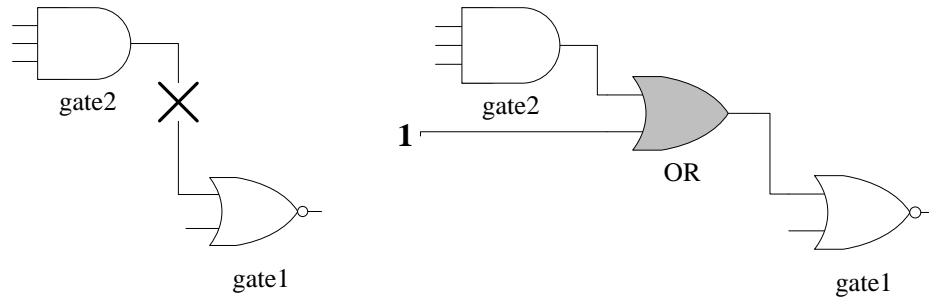
### 5.3 Assumptions

There are few assumptions made in our reliability evaluator as follows:

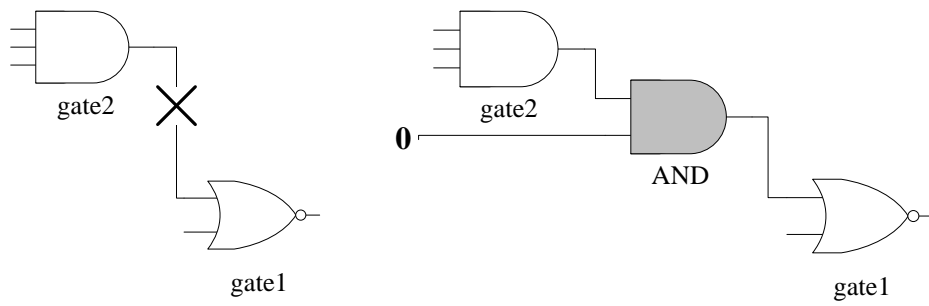
- As we are experimenting the effect of soft errors, the number of injected faults should be small. In reality, they are measured by FIT (failure in each  $10^9$  hours of operation).
- The number of iterations,  $T$ , is 10000.
- The faults are injected randomly at the gate level of a circuit.
- Only logical masking is considered by our evaluator. Neither electrical masking nor timing masking is considered.

### 5.4 Fault Model and Fault Injection Mechanism

In our work, we assume a stuck at fault model. When we inject faults at any line, it can be either stuck-at-1 (i.e. connected to Vdd) or stuck-at-0 (i.e. connected to ground). Fault injection happens at the gate level of the circuit randomly, a list of all lines in a circuit is constructed and at each time a random line is selected. A line can be a gate input, gate output, or a fan-out line. In case of a stuck-at-1 fault, the line at which the fault is injected is replaced by an OR-gate with a fault indicator input  $F_i$  set to 1. When a stuck-at-0 fault is injected, the line at which the fault is injected is replaced by an AND-gate with a fault indicator input  $F_i$  set to 0. This is shown in Figure 5.1.



**Stuck at 1**



**Stuck at 0**

Figure 5.1: Fault injection mechanism.

## Chapter 6

# **RESULTS, DISCUSSION AND FINDINGS**

In this chapter, we will discuss the results for some experiments that have been performed. First, we will describe the experiment we choose to perform and the justification for such choice. Then, a detailed discussion of the findings is given for Algorithm 1 described in Chapter 3 and Algorithm 2 described in Chapter 4.

### **6.1 Experiments**

Several experiments have been performed for different MCNC combinational circuits. The circuits m3, bench1, test1, test4, ex1010, apex3, apex4, exp, misex3, cps, spla, duke2, table3 and table5 are the ones chosen for our experiments. Table 6.1 shows the number of primary inputs, primary outputs, and the area for each benchmark circuit. Section 6.2 explains how area is calculated for benchmark circuits.

For each circuit, several versions are implemented and experimented. The original version was obtained by synthesizing the pla circuit using SIS tool to get a multi-level

circuit. This is done using “*script.rugged*” script. Then, different versions were obtained by applying the framework discussed in Chapter 4.

Table 6.1: primary inputs, primary outputs and area for benchmark circuits.

Circuits	Primary Inputs	Primary Outputs	Original
<b>apex4</b>	9	18	3010
<b>bench1</b>	9	9	1487
<b>cps</b>	24	102	1728
<b>duke2</b>	22	29	632
<b>ex1010</b>	10	10	4712
<b>exp</b>	8	18	430
<b>m3</b>	8	16	358
<b>misex3</b>	14	14	1020
<b>spla</b>	16	46	555
<b>table3</b>	14	14	1171
<b>table5</b>	17	15	1333
<b>test1</b>	8	10	1189
<b>test4</b>	8	30	3083
<b>apex3</b>	54	50	2302

Several experiments to test Algorithm 1 described in Chapter 3 are implemented and experimented as follows:

- Original circuit: in this version, circuit is implemented as a multi-level network.
- Espresso version: in this version, the extracted sub-circuits are synthesized using Espresso tool, the phase (on or off) with the higher probability is selected to implement the sub-circuit.

- Irredundant Two-level synthesis: extracted sub-circuits are synthesized using the proposed two-level synthesis algorithm (Algorithm 1), such that the cover of the sub-circuit contains irredundant cubes. The phase with higher probability is implemented. No duplication is used in this version.
- Redundant threshold 0.05: two-level synthesis algorithm (Algorithm 1) is used with redundant cover. The redundancy threshold is 0.05. The phase with higher probability is implemented. No duplication is used in this version.
- Redundant threshold 0.01: Algorithm 1 is used with redundant cover. The redundancy threshold is 0.01. No duplication is used in this version.
- Redundant threshold 0.005: Algorithm 1 is used with redundant cover. The redundancy threshold is 0.005. No duplication is used in this version.
- Redundant threshold 0.01 - Duplication: Algorithm 1 is used with redundant cover. The redundancy threshold is 0.01. *Duplication* is used in this version.

To test Algorithm 2, the following experiments are implemented:

- Original circuit: in this version, circuit is implemented as a multi-level network.
- Algorithm 1 - threshold 0.01: Algorithm 1 is used with redundant cover. The redundancy threshold is 0.01. Duplication is used.
- Fx for area: original version of fast extraction (fx) is applied on the sub-circuits used in the experiment Algorithm 1 - threshold 0.01.

- Algorithm 2 with 0.3-threshold: Algorithm 2 is applied on the sub-circuits used in the experiment (Algorithm 1 - threshold 0.01) with a threshold of 0.3.
- Algorithm 2 with 0.1-threshold: Algorithm 2 is applied on the sub-circuits used in the experiment (Algorithm 1 - threshold 0.01) with a threshold of 0.1.

Each of the experiments for each circuit follows the framework of Figure 6.1 (In Algorithm 1 experiments, Algorithm 2 phase is not applied). The original benchmark circuit in pla format is taken and converted into a multi-level circuit. Then, sub-circuits are extracted along with their windows. The corresponding window circuit for each extracted circuit will be used to find the Controllability Don't Care conditions (CDC's). If the number of primary inputs of the circuit is feasible (less than 25) the window circuit for each extracted sub-circuit will start from the output of the sub-circuit and go back traversal until reaching the primary inputs of the circuit. In such case, all possible don't care inputs will be calculated. If the number of primary inputs is greater than or equal 25 the number of window inputs will be limited to 20. Next, the On and Off minterms are found for each extracted sub-circuit. The probabilities of occurrence for all On and Off minterms are also found using a developed simulator. After that, Algorithm 1 is applied on each sub-circuit taking On and Off minterms along with their probabilities as arguments. Algorithm 2 can be applied on each sub-circuit resulting after applying Algorithm 1. Finally, the re-synthesized sub-circuits are merged back to the original circuit. Ultimately, the new re-synthesized circuit is accepted as an argument by our evaluator described in Section 5.2.

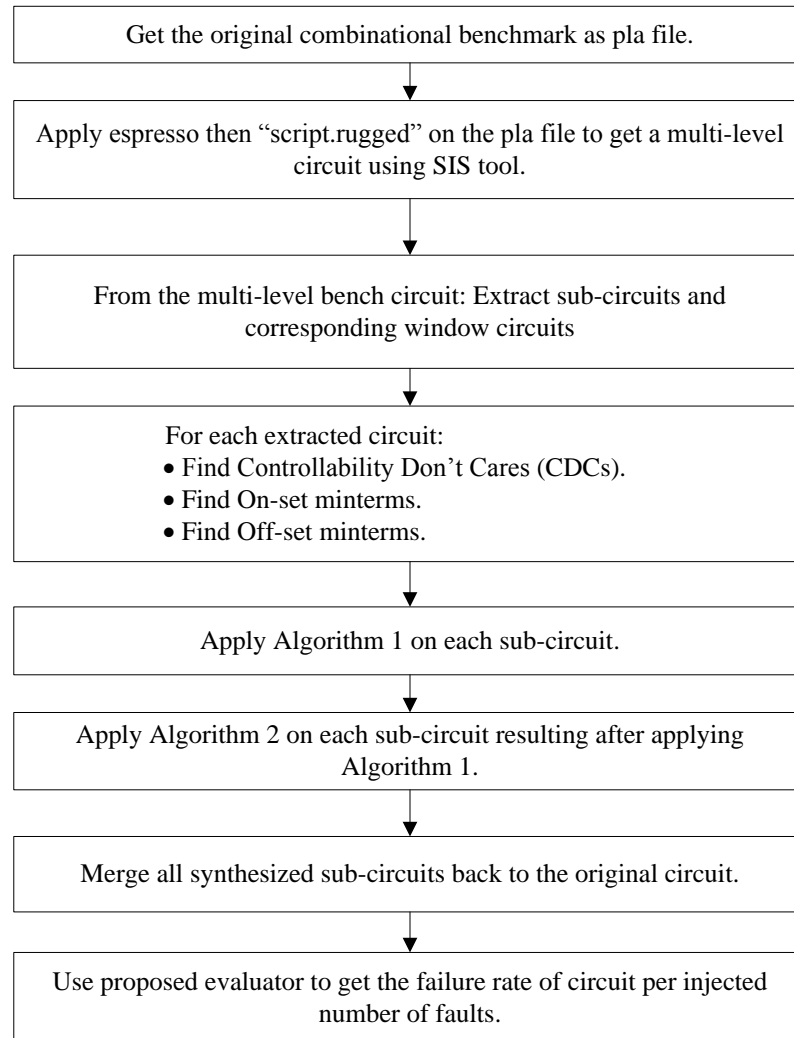


Figure 6.1: Framework of experiments.



## 6.2 Calculating the Area Overhead

Techniques involving adding redundancy to enhance reliability often pay the price in terms of area. To estimate the area, each resulting circuit is mapped to the `mcnc.genlib` library. The total size of gates is found using SIS tool using the command “`map -m 0`”. Table 6.2 shows the used library to estimate the size of circuits.

Table 6.2: Size of gates.

Gate	Size
NOT	1
NAND2	2
NAND3	3
NAND4	4
NOR2	2
NOR3	3
NOR4	4
AND2	3
OR2	3
XOR	5
XNOR	5
AOI21	3
AOI22	4
OAI21	3
OAI22	4

## 6.3 Algorithm 1 Results

In this section, we will apply different versions of Algorithm 1 to the circuits as stated in Section 6.1. The resulting findings are discussed particularly for `bench1` and for the remaining circuits in the next few sections.

### 6.3.1 Case Study: Bench1 Benchmark

The combinational circuit bench1 is picked to discuss general results. Figure 6.2 shows the failure rate results for the different versions specified in Section 6.1. It shows the failure rate of bench1 circuit by applying Algorithm1. It is compared to the original circuit. Also it is compared to circuit failure rate after applying espresso on extracted sub-circuits. It's clear that applying espresso on the extracted sub-circuits to get a two-level implementation results in a more reliable circuit compared to the original version. This is due to synthesizing the phase with higher probability of occurrence. However, Algorithm 1 results outperform espresso results since we carefully select the minimum cover which maximizes fault masking, and then we add extra cubes to the cover to mask faults for not masked minterms.

If the results of non-redundant version are compared with the espresso version results, a significant improvement can be noticed. In the non-redundant version, the minimum sets of cubes are selected to maximize fault masking. Extra cubes can be added to the cover in the redundant versions, such that an additional improvement on fault masking is obtained. This is controlled by the threshold. The price of this improvement is a larger area. The redundant versions show that as we decrease the threshold value, we obtain better failure rate. The 0.05 threshold does not improve failure rate compared to the irredundant version. However, the 0.01 value improves the failure rate significantly. The 0.005 value also adds more improvement. The last version which includes duplication improves failure rate significantly compared to *Redundant threshold 0.01* version. This is due to the significant number of sub-circuits that were duplicated.

To complete the picture, Figure 6.3 compares the area of the implementation of the discussed curves. It shows that the area of the implemented circuit increases as we add redundant cubes to the cover in the redundant versions, which is a logical expected price paid to enhance fault masking. Espresso version adds an area overhead of 26% compared to the original circuit. The irredundant version overhead is 37%. The area overhead of the redundant version with thresholds 0.05 is very close to the irredundant version. The 0.01 version adds a significant area overhead of 60%. The 0.005 version also adds more area overhead of 76%. However, the overhead of duplication using 0.01 is 76%. Using duplication in this case has significant increase in area overhead compared to the version with no duplication, and this explains the significant improvement obtained using duplication. In fact, duplication effect depends on the benchmark circuit itself. It depends on how many generated sub-circuits with only one cube.

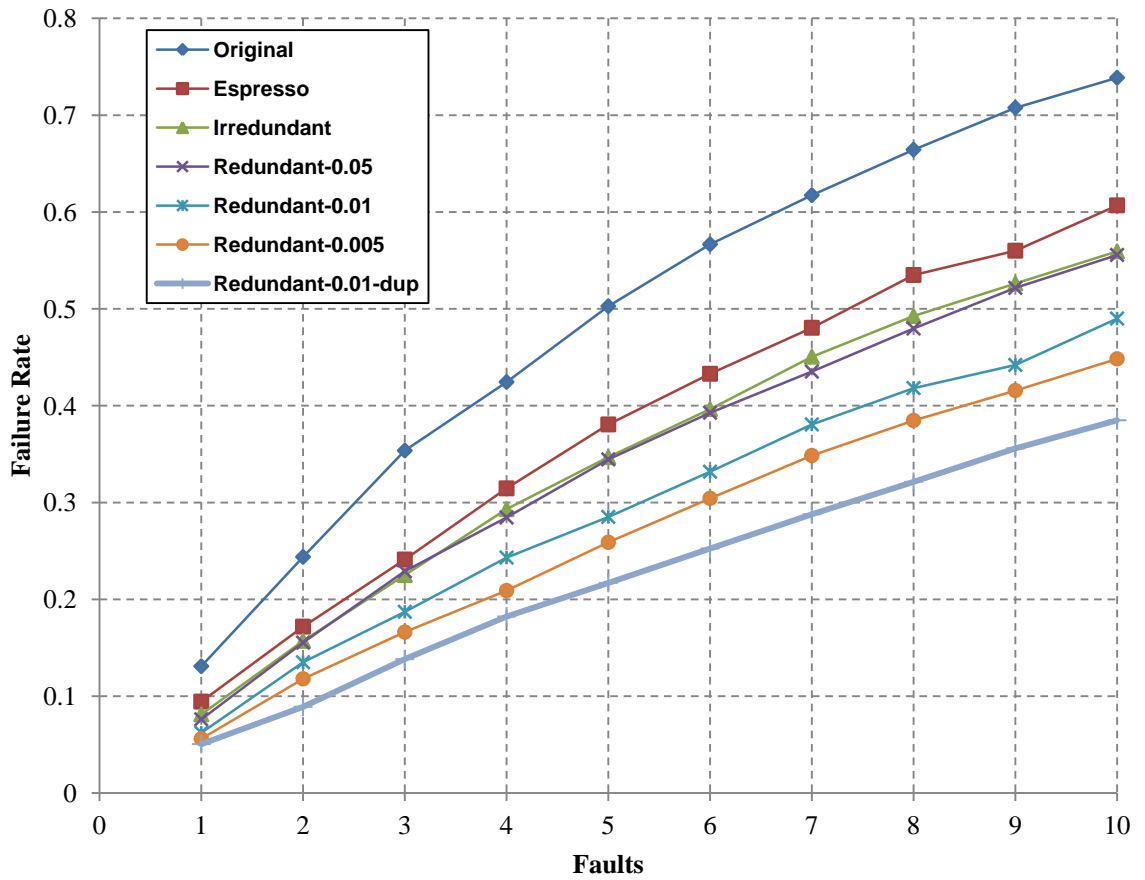


Figure 6.2: Failure rate vs Faults for bench1.

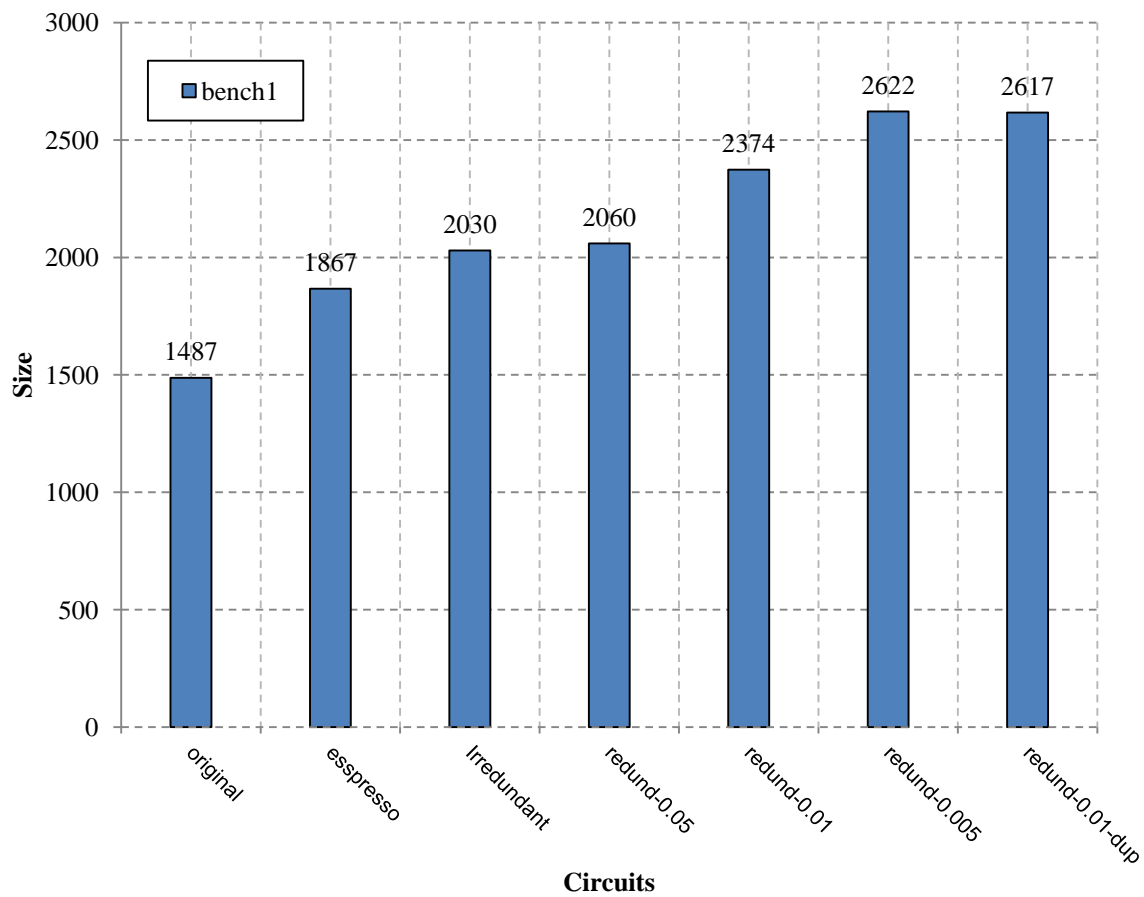


Figure 6.3: Size of bench1 experiments.

### 6.3.2 Other Benchmarks Results

Simulation results illustrating failure rate for other benchmark circuits are given from Figure 6.4 to Figure 6.16. The failure rate curves for all versions are shown except for the version with threshold 0.05, since its results are very close to the irredundant version results. The results for 0.05-threshold version are shown in the next sub-section in the aggregated tables.

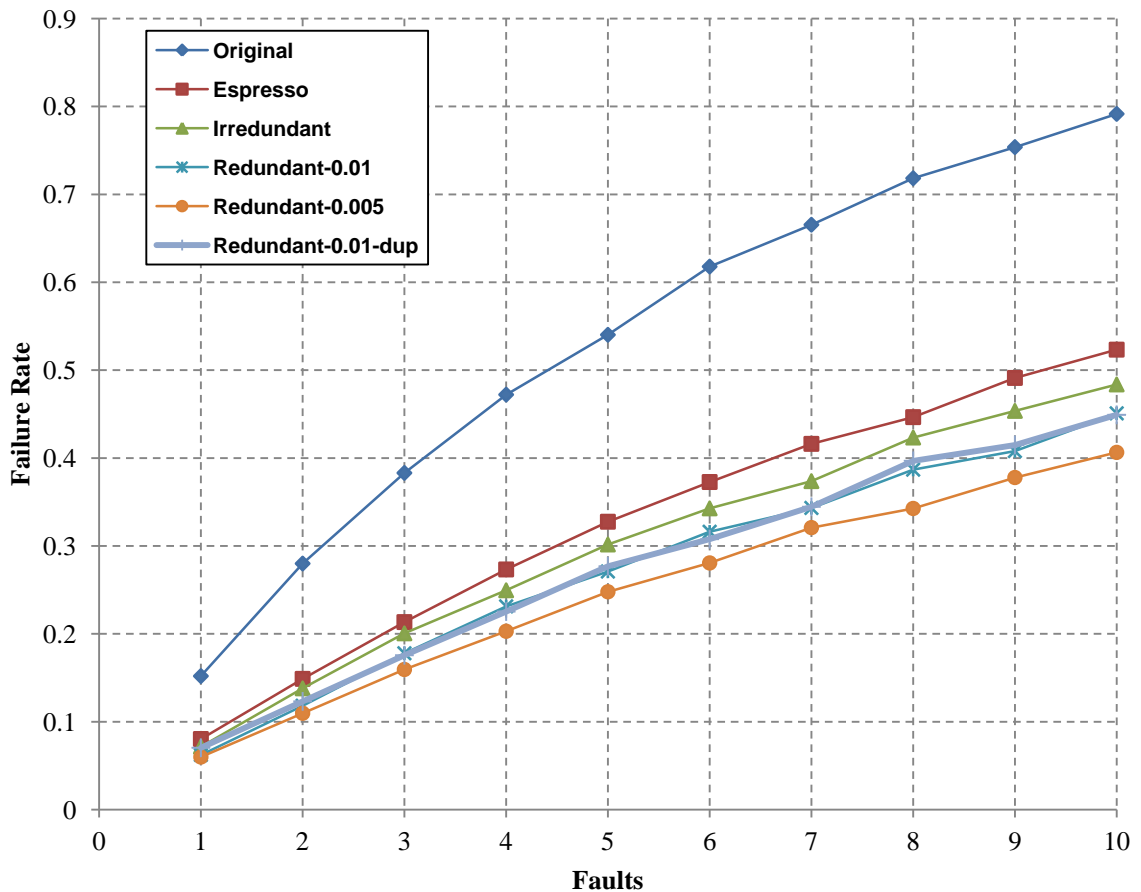


Figure 6.4: Failure rate vs Faults for m3.

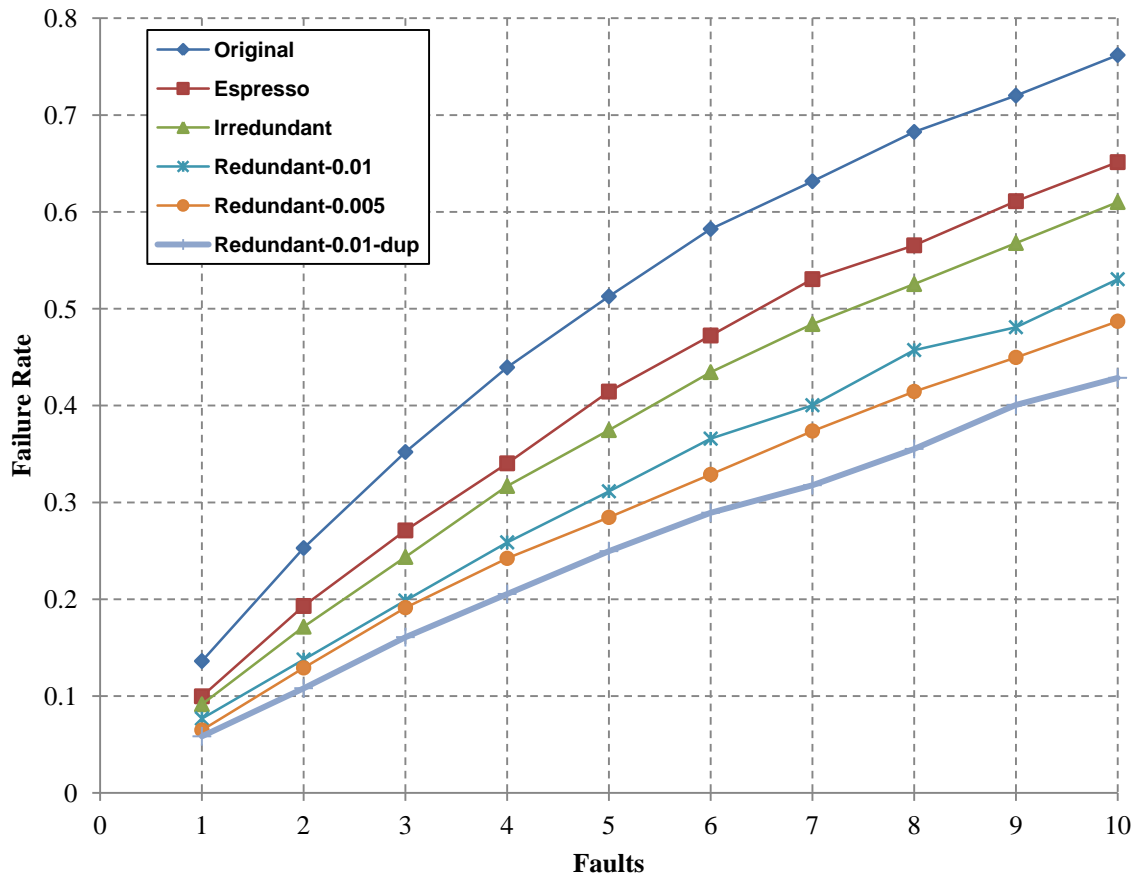


Figure 6.5: Failure rate vs Faults for test1.

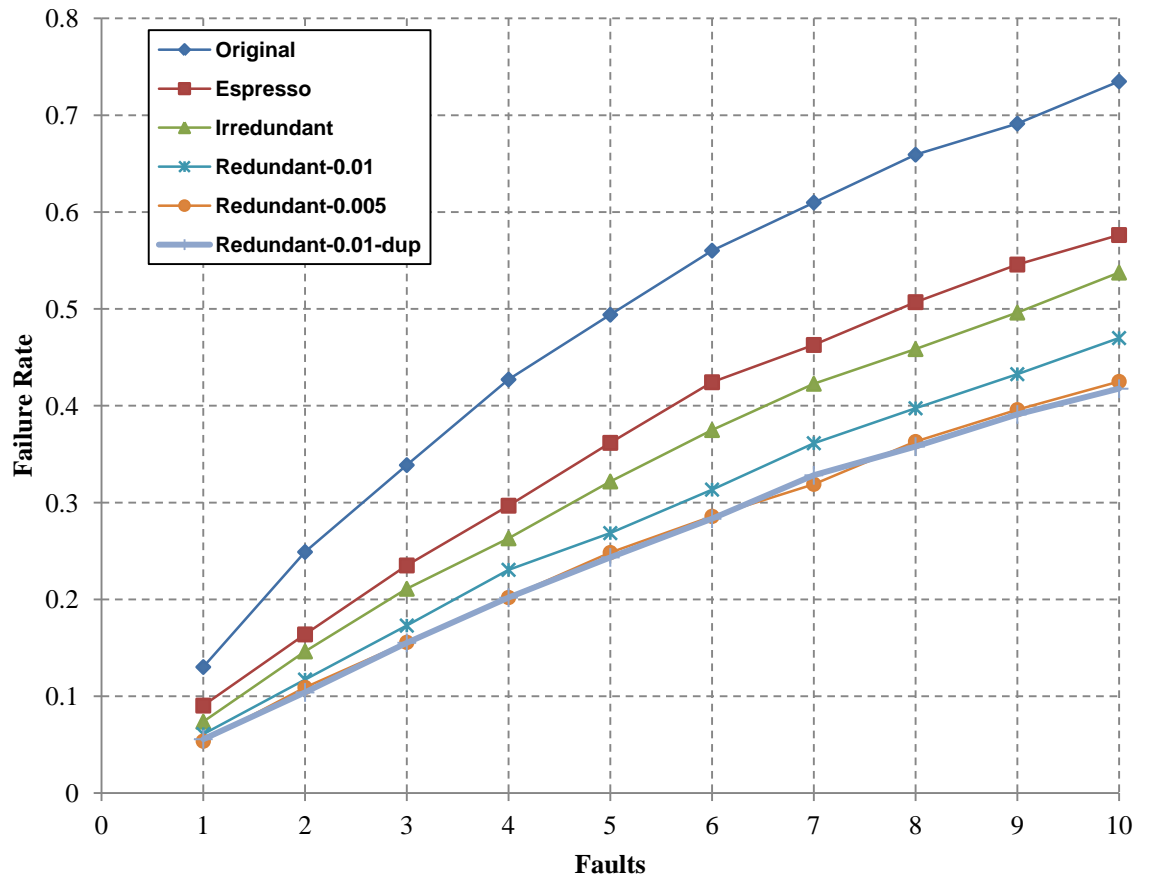


Figure 6.6: Failure rate vs Faults for test4.



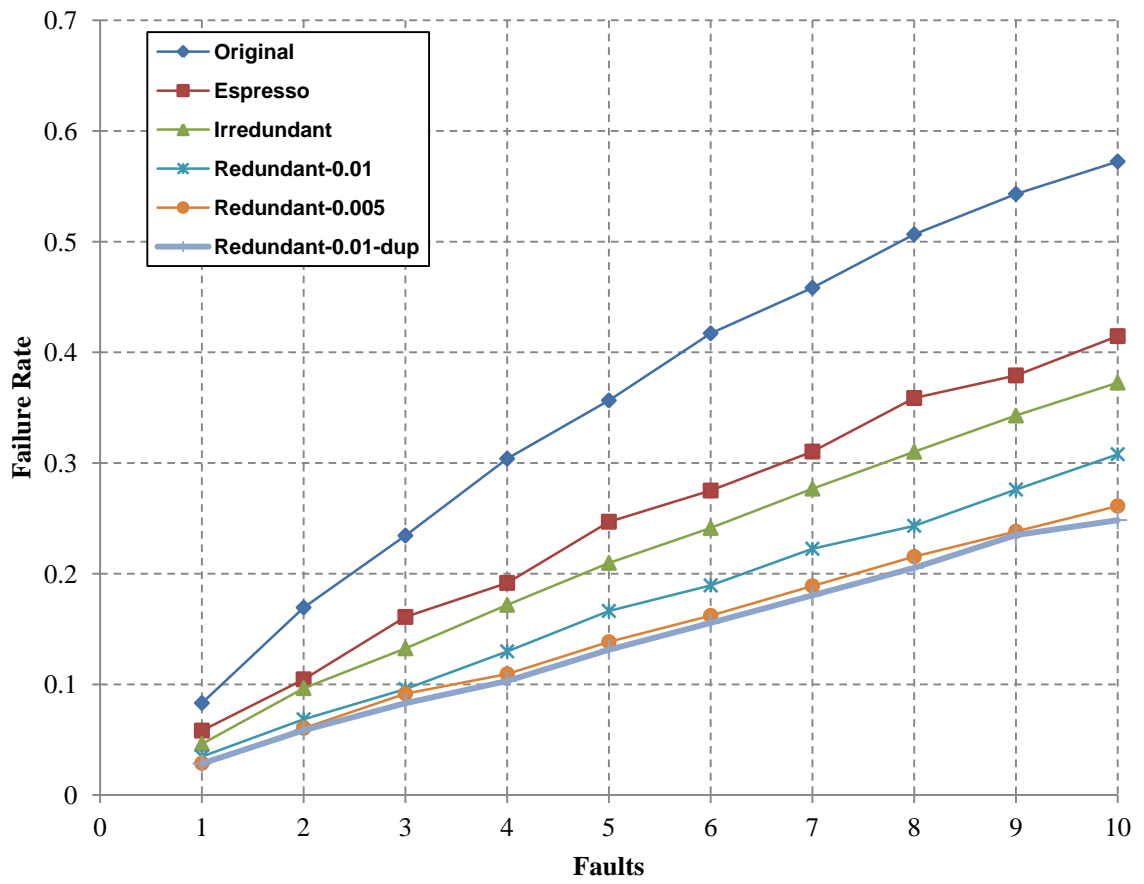


Figure 6.7: Failure rate vs Faults for ex1010.

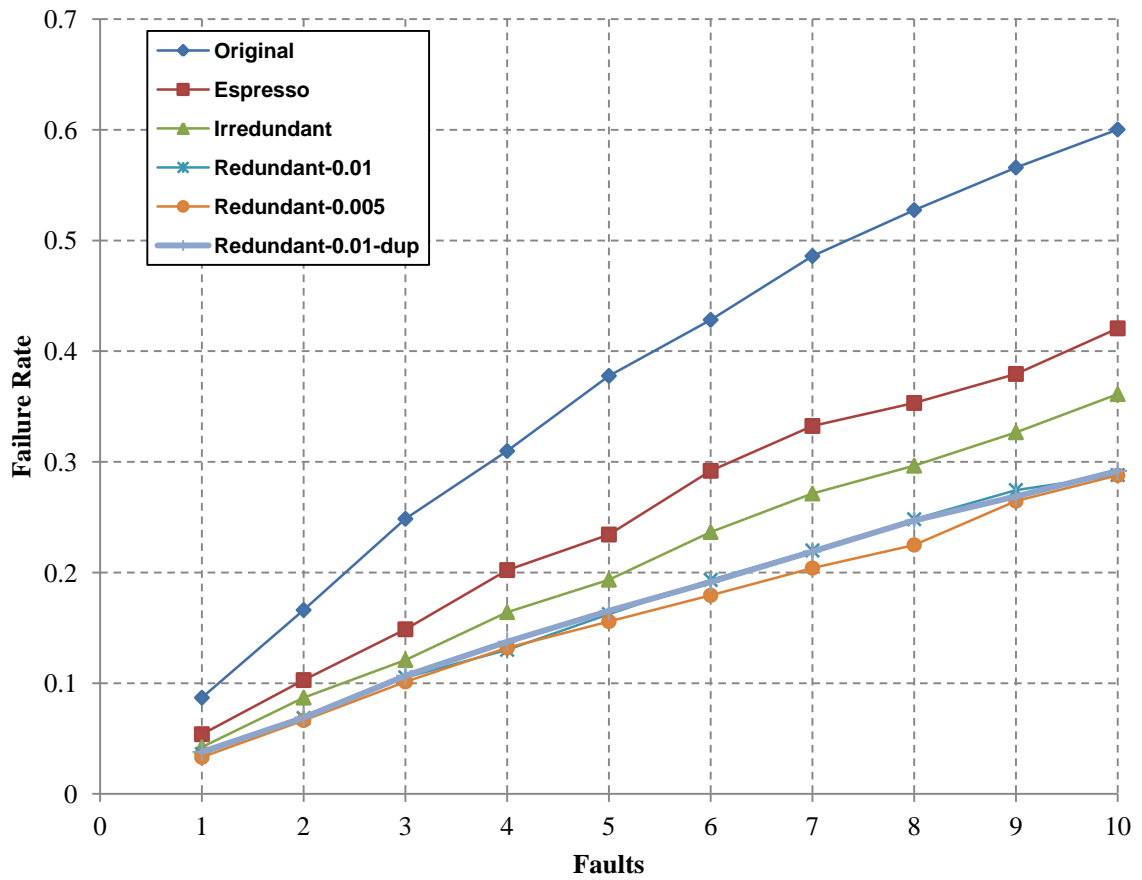


Figure 6.8: Failure rate vs Faults for misex3.

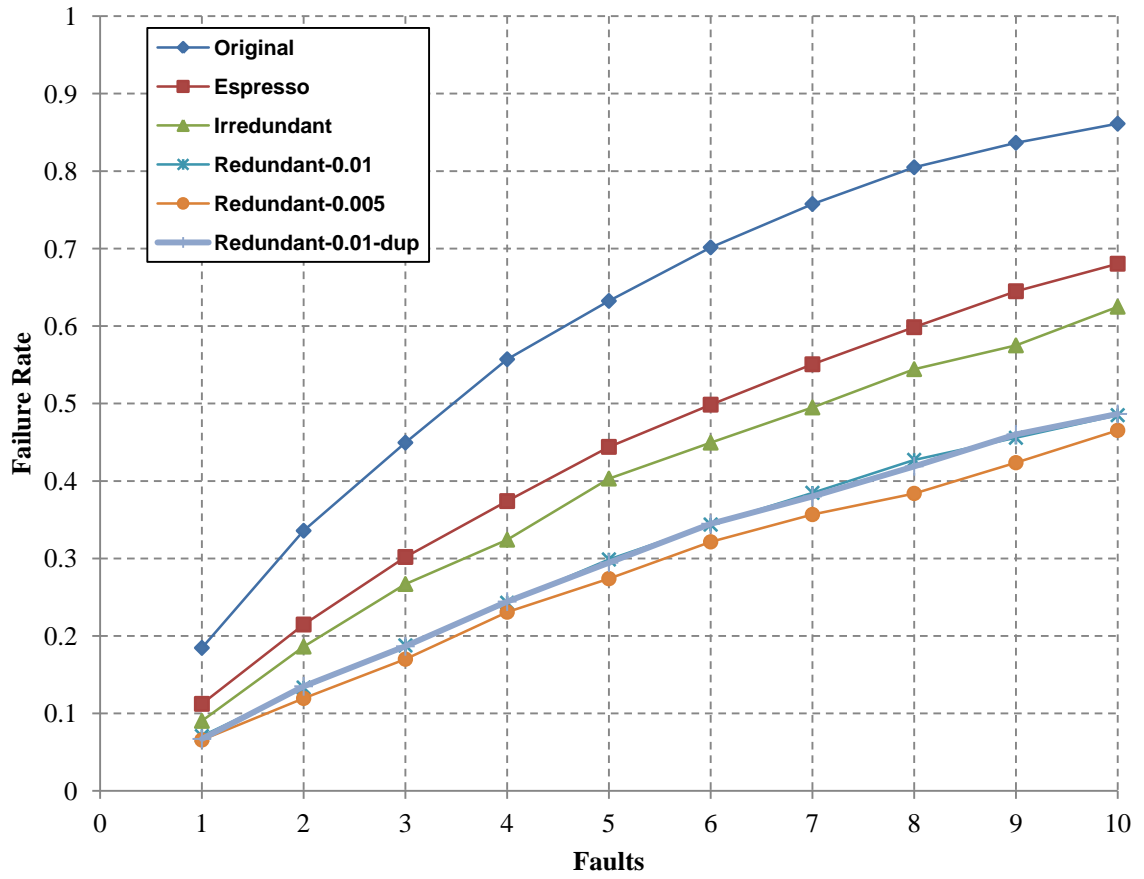


Figure 6.9: Failure rate vs Faults for exp.

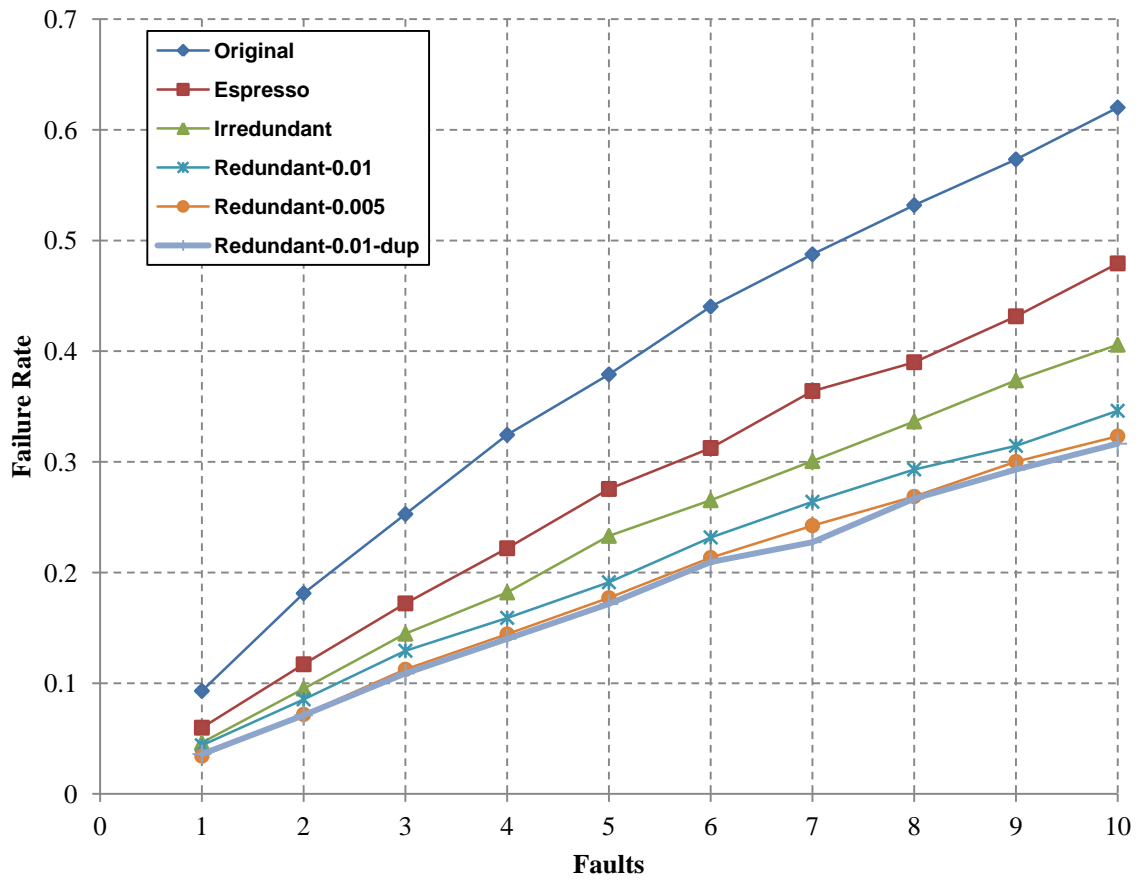


Figure 6.10: Failure rate vs Faults for apex4.

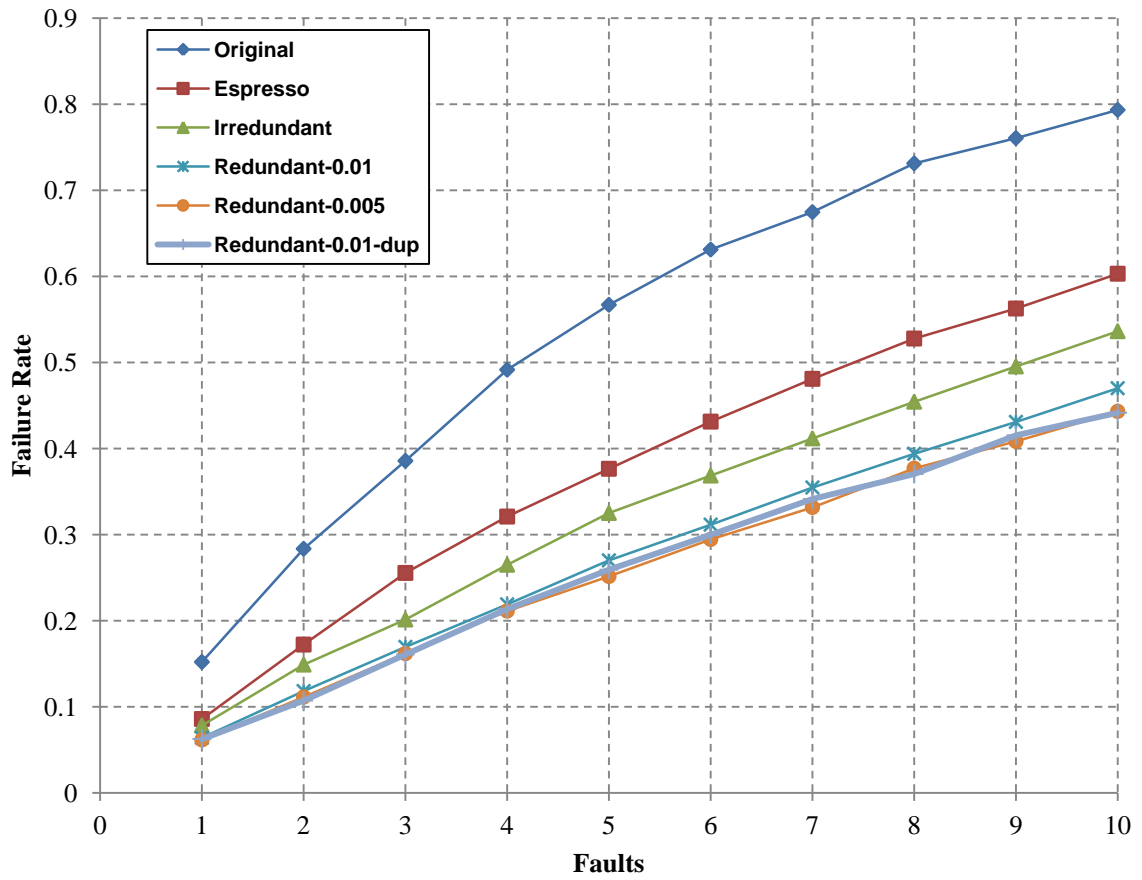


Figure 6.11: Failure rate vs Faults for duke2.

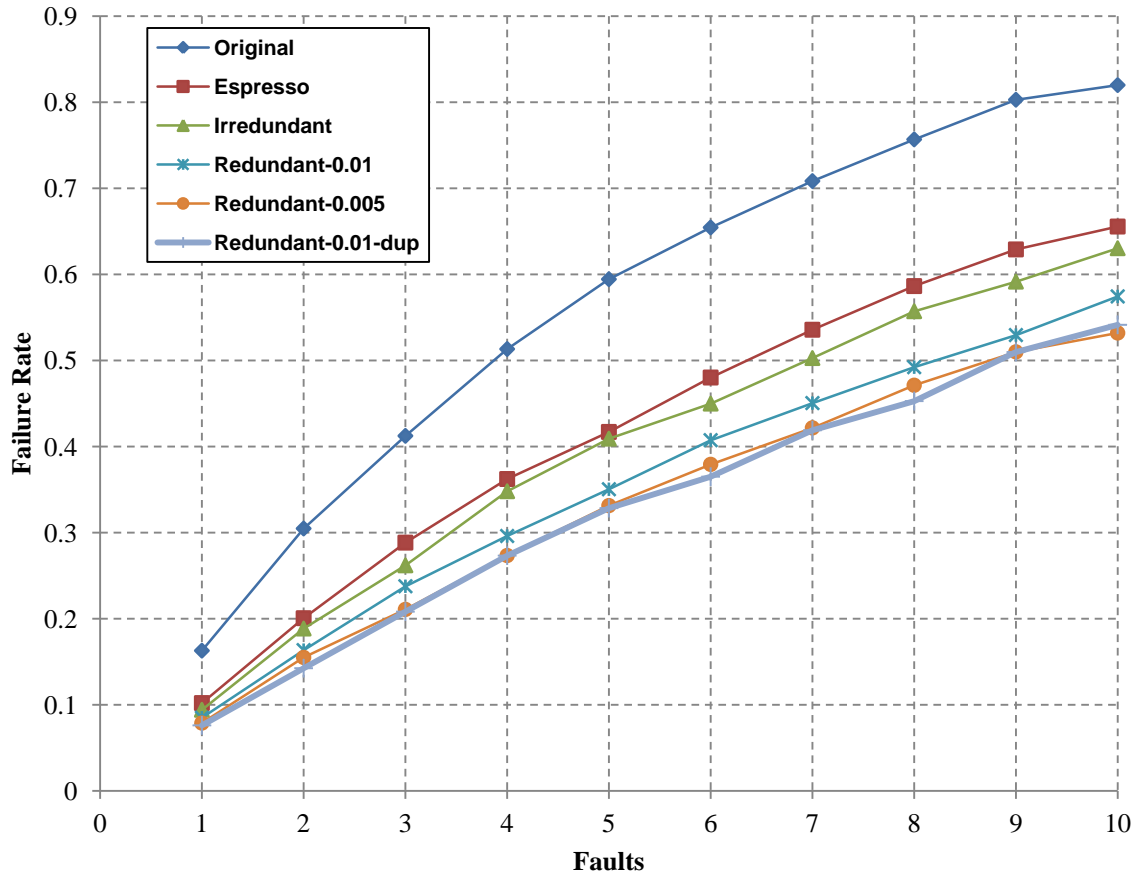


Figure 6.12: Failure rate vs Faults for spla.

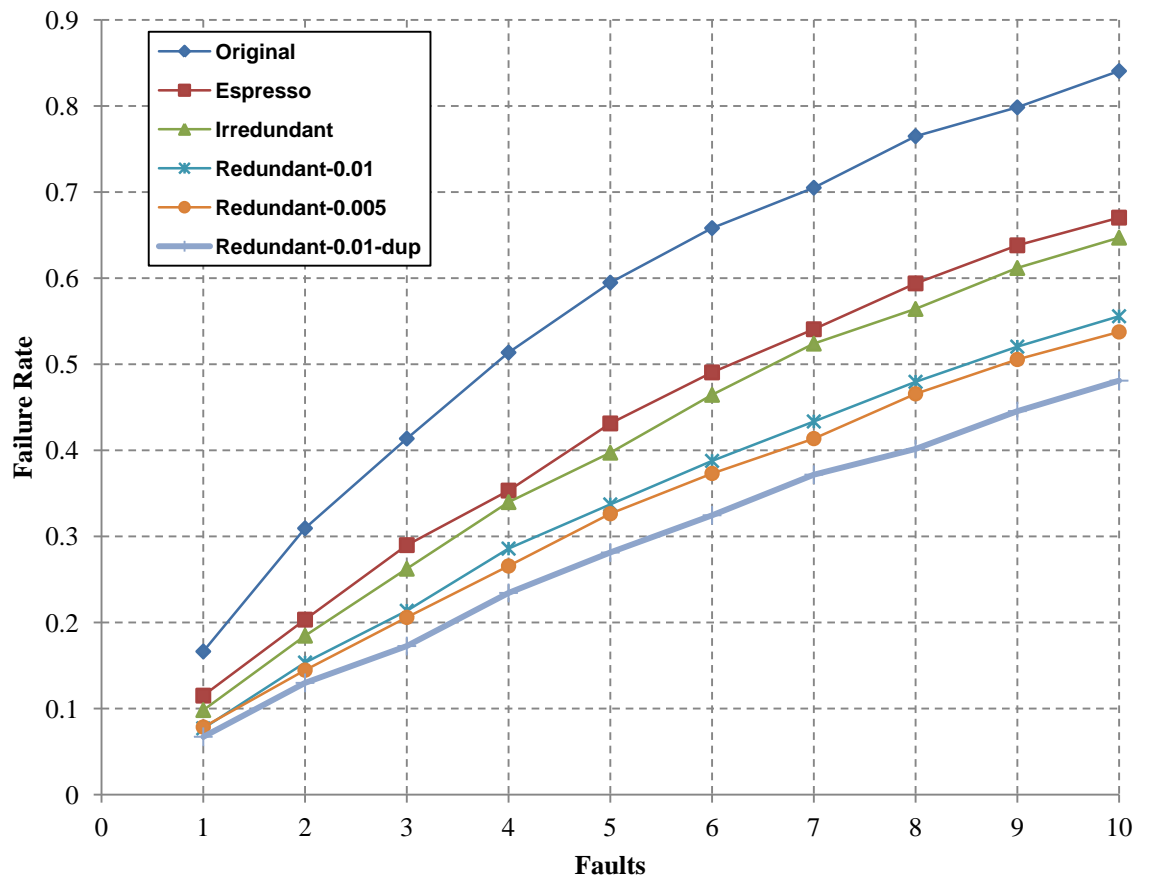


Figure 6.13: Failure rate vs Faults for cps.

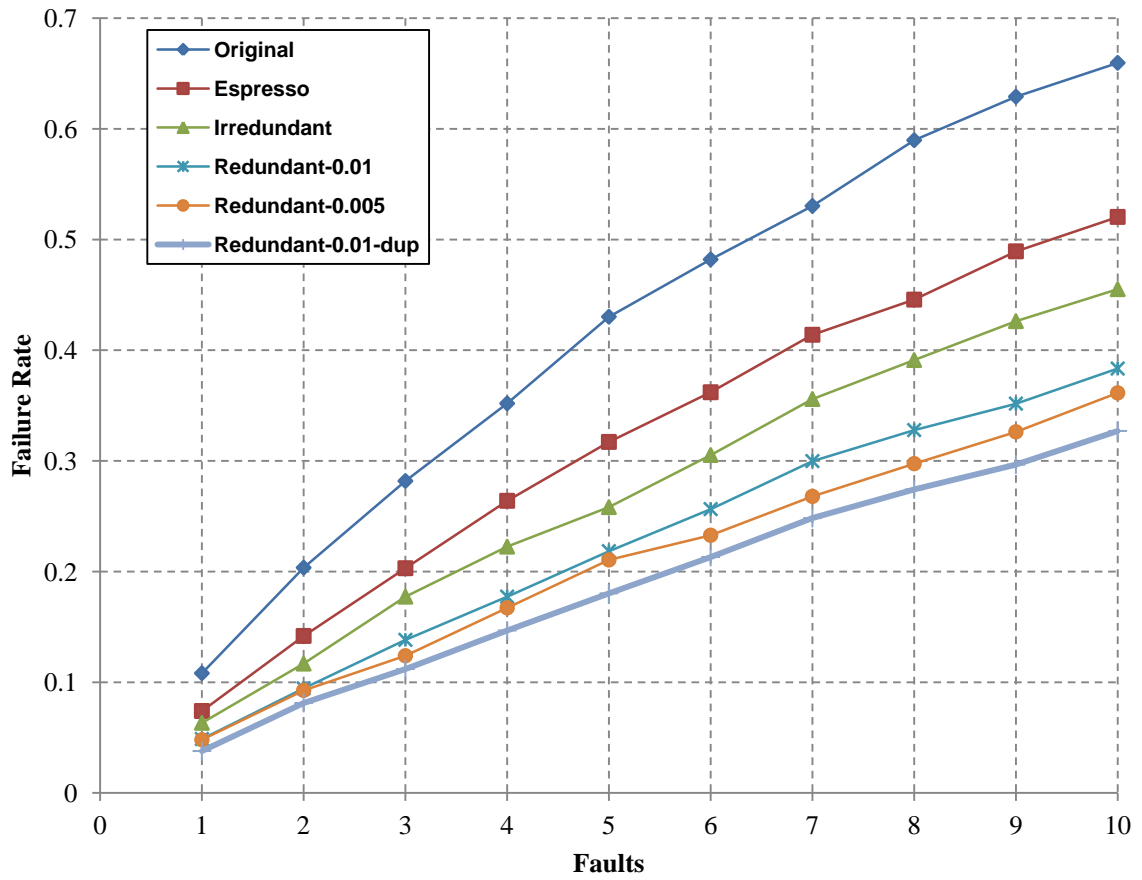


Figure 6.14: Failure rate vs Faults for table3.



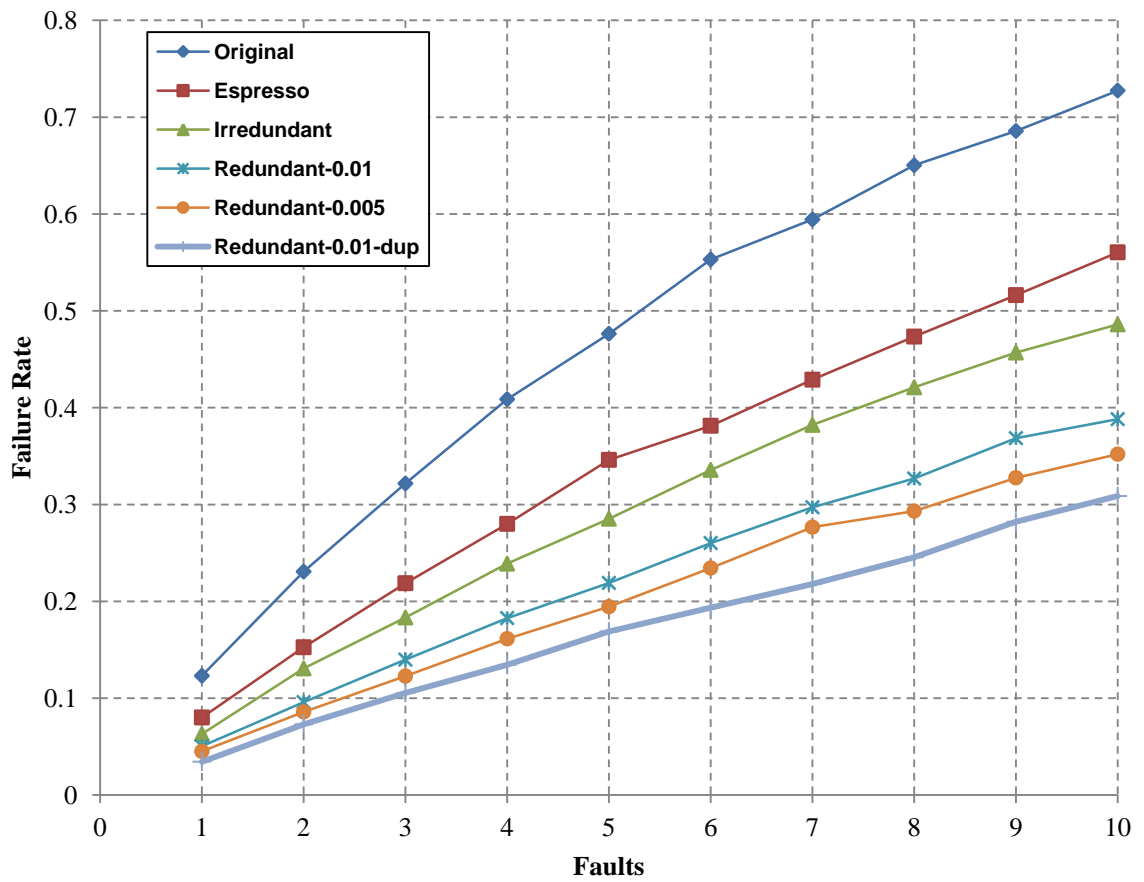


Figure 6.15: Failure rate vs Faults for table5.

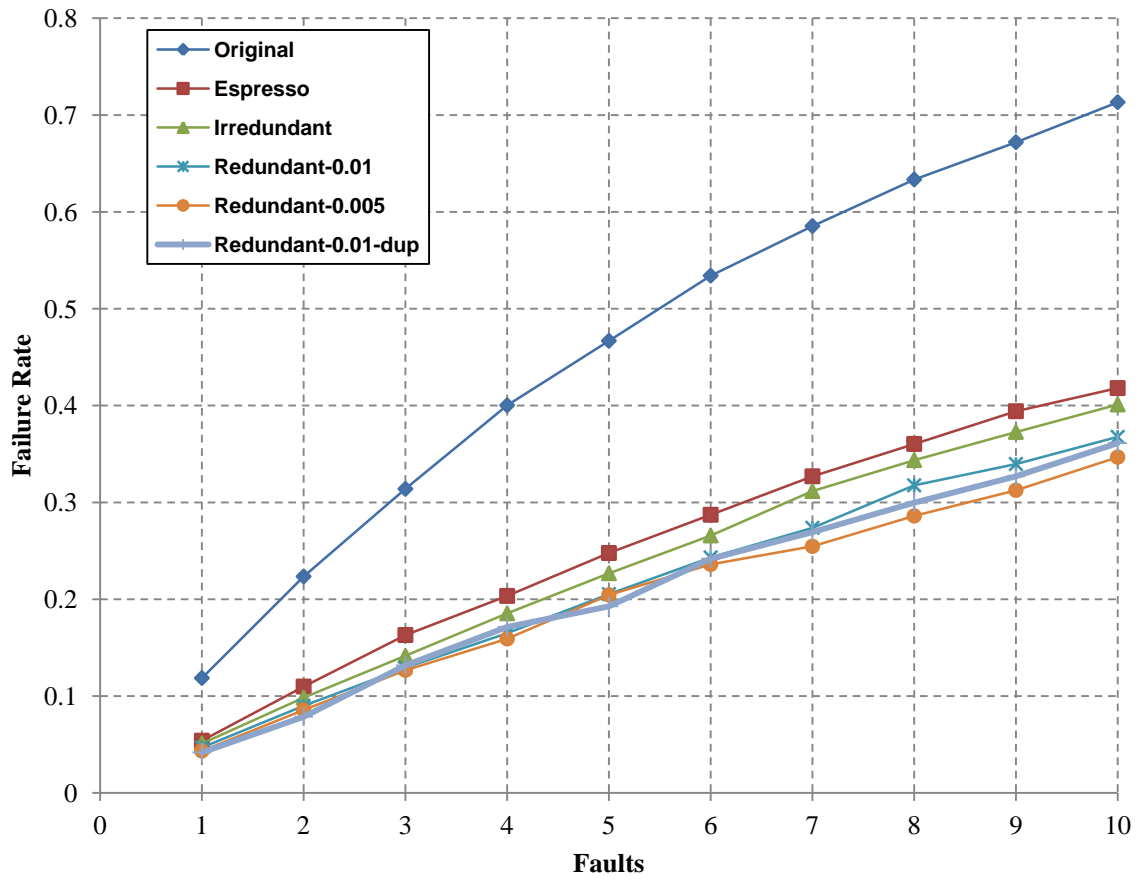


Figure 6.16: Failure rate vs Faults for apex3.

As shown in the previous results, using Algorithm1 with different versions provides significant improvement in term of failure rate. Each version improves the failure rate significantly compared to Espresso version. However, the apex3 benchmark experiments didn't show a significant improvement compared to Espresso version. This is due to the large number of primary inputs of this benchmark. This causes the size of window circuits corresponding to extracted sub-circuits to be limited to a certain depth in the original circuit, and not reaching the primary inputs of the circuit. And so, not all possible don't cares will be calculated. This will limit the flexibility to select between different covering cubes to cover a sub-circuit minterms.

### **6.3.3 Aggregated Results and Conclusions**

In this sub-section, we report the aggregated results from combinational benchmark circuits using Algorithm 1. Results of failure rate for benchmark circuits by injecting 1, 5, and 10 faults are shown in Table 6.3, Table 6.4 and Table 6.5. In those tables, the amount of reduction (-ve sign in the table) compared to the original circuits in terms of failure rate is stated and averaged among all circuits. For each circuit, the failure rate of all Algorithm 1 experiments explained in section 6.1 is reported.

From these tables, we can draw the following conclusions:

- By injecting 1 fault, the best failure rate reduction without using duplication is 60% for 0.005 threshold. This is followed by 55% failure rate reduction for 0.01 threshold, 49% failure rate reduction for 0.05 threshold, 46% failure rate for irredundant version, then 36% failure rate reduction for Espresso version. The failure rate reduction using duplication with 0.01 threshold is 62%.

- By injecting 5 faults, the best failure rate reduction without using duplication is 53% for 0.005 threshold using Algorithm 1. This is followed by 49% failure rate reduction for 0.01 threshold, 41% failure rate reduction for 0.05 threshold, 39% failure rate for irredundant version, then 30% failure rate reduction for Espresso version. The failure rate reduction using duplication with 0.01 threshold is 56%.
- By injecting 10 faults, the best failure rate reduction without using duplication is 45% for 0.005 threshold using Algorithm 1. This is followed by 41% failure rate reduction for 0.01 threshold, 33% failure rate reduction for 0.05 threshold, 31% failure rate for irredundant version, then 24% failure rate reduction for Espresso version. The failure rate reduction using duplication with 0.01 threshold is 48%.

The area for each experiment in each circuit is shown in Table 6.6. The most area overhead on average compared to the original area is 86% for 0.005 threshold. This is followed by 79% for 0.01 threshold with duplication, 70% for 0.01 threshold without duplication, 50% for 0.05 threshold, 47% for irredundant version, and then 38% for Espresso version.

The overall averages derived from the previously mentioned tables are shown in Table 6.7. We can say that on average, the best failure rate reduction is 55% for the 0.01 duplicate version. This is followed by 52% for 0.005 threshold, 48% for 0.01 threshold, 41% for 0.05 threshold, 38% for irredundant version, and then 30% for Espresso version.

Based on the results in this sub-section, we can draw the following conclusions:

- As we decrease the value of redundancy threshold in Algorithm 1, as we obtain a better circuit in terms of failure rate, but with a larger cost in terms of area overhead.
- Effectiveness of duplication depends on the circuit itself. Some circuits show a significant improvement using duplication like test1, bench1, cps, table3 and table5 circuits.
- Using Redundancy threshold of 0.01 with duplication is recommended since it reduces the failure rate dramatically by 55% on average when compared to original circuit. However, the area overhead is 79% on average.
- Larger circuits in terms of area have better failure rates reductions, since the number of extracted circuits will be larger and there will be more room for maximizing fault masking.
- Circuits with large number of primary inputs have limited failure reductions, since we can't calculate all possible don't cares for extracted sub-circuits in a feasible time.

Table 6.3: Failure rate results for MCNC combinational benchmark circuits - Injecting 1 fault.

Circuits	Original	Comparison Espresso	Algorithm 1 - Redundancy Threshold				
			Irredundant	0.05	0.01	0.005	0.01-Duplicate
apex3	0.119	0.054	0.051	0.051	0.047	0.043	0.042
apex4	0.093	0.060	0.046	0.045	0.044	0.034	0.036
bench1	0.131	0.094	0.081	0.076	0.062	0.056	0.051
cps	0.166	0.115	0.098	0.089	0.077	0.070	0.067
duke2	0.152	0.086	0.078	0.070	0.064	0.062	0.063
ex1010	0.083	0.058	0.046	0.042	0.035	0.029	0.028
exp	0.185	0.112	0.090	0.087	0.071	0.066	0.067
m3	0.152	0.080	0.072	0.071	0.062	0.060	0.059
misex3	0.087	0.054	0.042	0.040	0.036	0.033	0.034
spla	0.163	0.102	0.094	0.090	0.085	0.079	0.076
table3	0.108	0.074	0.063	0.061	0.049	0.048	0.038
table5	0.123	0.080	0.063	0.063	0.050	0.045	0.034
test1	0.136	0.100	0.092	0.087	0.077	0.065	0.058
test4	0.130	0.090	0.074	0.071	0.061	0.054	0.056

Circuits	Original	Comparison Espresso	Algorithm 1 - Redundancy Threshold				
			Irredundant	0.05	0.01	0.005	0.01-Duplicate
apex3	0%	-55%	-57%	-57%	-61%	-63%	-65%
apex4	0%	-36%	-50%	-52%	-53%	-63%	-61%
bench1	0%	-28%	-38%	-42%	-52%	-57%	-61%
cps	0%	-31%	-41%	-46%	-54%	-58%	-60%
duke2	0%	-44%	-49%	-54%	-58%	-60%	-59%
ex1010	0%	-30%	-44%	-49%	-58%	-65%	-66%
exp	0%	-39%	-51%	-53%	-62%	-64%	-64%
m3	0%	-47%	-53%	-54%	-59%	-61%	-61%
misex3	0%	-38%	-51%	-54%	-58%	-62%	-61%
spla	0%	-37%	-42%	-45%	-48%	-52%	-53%
table3	0%	-32%	-41%	-44%	-55%	-56%	-65%
table5	0%	-35%	-49%	-49%	-59%	-63%	-72%
test1	0%	-27%	-33%	-36%	-44%	-52%	-57%
test4	0%	-31%	-43%	-46%	-53%	-59%	-57%
<b>Average</b>	0%	-36%	-46%	-49%	-55%	-60%	-62%

Table 6.4: Failure rate results for MCNC combinational benchmark circuits - Injecting 5 faults.

Circuits	Original	Comparison Espresso	Algorithm 1 - Redundancy Threshold				
			Irredundant	0.05	0.01	0.005	0.01-Duplicate
apex3	0.467	0.248	0.227	0.228	0.205	0.204	0.193
apex4	0.379	0.276	0.233	0.226	0.191	0.177	0.169
bench1	0.503	0.381	0.347	0.345	0.285	0.259	0.208
cps	0.595	0.431	0.397	0.372	0.337	0.326	0.281
duke2	0.567	0.377	0.325	0.312	0.270	0.252	0.259
ex1010	0.357	0.247	0.210	0.198	0.166	0.138	0.130
exp	0.632	0.444	0.403	0.368	0.299	0.274	0.294
m3	0.540	0.328	0.302	0.294	0.271	0.248	0.267
misex3	0.378	0.234	0.194	0.192	0.162	0.156	0.160
spla	0.594	0.417	0.409	0.395	0.350	0.331	0.329
table3	0.430	0.317	0.258	0.252	0.218	0.211	0.180
table5	0.476	0.346	0.285	0.262	0.219	0.195	0.169
test1	0.513	0.414	0.375	0.370	0.311	0.285	0.235
test4	0.494	0.362	0.322	0.303	0.269	0.248	0.243

Circuits	Original	Comparison Espresso	Algorithm 1 - Redundancy Threshold				
			Irredundant	0.05	0.01	0.005	0.01-Duplicate
apex3	0%	-47%	-51%	-51%	-56%	-56%	-59%
apex4	0%	-27%	-38%	-40%	-50%	-53%	-55%
bench1	0%	-24%	-31%	-31%	-43%	-48%	-59%
cps	0%	-28%	-33%	-38%	-43%	-45%	-53%
duke2	0%	-34%	-43%	-45%	-52%	-56%	-54%
ex1010	0%	-31%	-41%	-45%	-53%	-61%	-64%
exp	0%	-30%	-36%	-42%	-53%	-57%	-54%
m3	0%	-39%	-44%	-46%	-50%	-54%	-51%
misex3	0%	-38%	-49%	-49%	-57%	-59%	-58%
spla	0%	-30%	-31%	-34%	-41%	-44%	-45%
table3	0%	-26%	-40%	-41%	-49%	-51%	-58%
table5	0%	-27%	-40%	-45%	-54%	-59%	-65%
test1	0%	-19%	-27%	-28%	-39%	-45%	-54%
test4	0%	-27%	-35%	-39%	-46%	-50%	-51%
Average	0%	-30%	-39%	-41%	-49%	-53%	-56%

Table 6.5: Failure rate results for MCNC combinational benchmark circuits - Injecting 10 faults.

Circuits	Original	Comparison Espresso	Algorithm 1 - Redundancy Threshold				
			Irredundant	0.05	0.01	0.005	0.01-Duplicate
apex3	0.713	0.418	0.401	0.403	0.368	0.347	0.340
apex4	0.620	0.479	0.406	0.398	0.346	0.323	0.304
bench1	0.739	0.607	0.559	0.556	0.490	0.448	0.367
cps	0.8407	0.6704	0.6471	0.6065	0.5558	0.5376	0.481
duke2	0.7934	0.6031	0.5363	0.5085	0.4701	0.4432	0.442
ex1010	0.572	0.415	0.373	0.349	0.308	0.261	0.247
exp	0.861	0.680	0.625	0.605	0.485	0.465	0.475
m3	0.792	0.523	0.484	0.484	0.451	0.406	0.435
misex3	0.600	0.421	0.361	0.342	0.288	0.288	0.284
spla	0.8199	0.6556	0.6304	0.6176	0.5744	0.532	0.541
table3	0.6596	0.5204	0.4551	0.4449	0.3834	0.3614	0.327
table5	0.7274	0.5604	0.486	0.4472	0.3881	0.352	0.309
test1	0.762	0.651	0.610	0.592	0.530	0.487	0.424
test4	0.735	0.576	0.538	0.524	0.470	0.425	0.418

Circuits	Original	Comparison Espresso	Algorithm 1 - Redundancy Threshold				
			Irredundant	0.05	0.01	0.005	0.01-Duplicate
apex3	0%	-41%	-44%	-44%	-48%	-51%	-52%
apex4	0%	-23%	-35%	-36%	-44%	-48%	-51%
bench1	0%	-18%	-24%	-25%	-34%	-39%	-50%
cps	0%	-20%	-23%	-28%	-34%	-36%	-43%
duke2	0%	-24%	-32%	-36%	-41%	-44%	-44%
ex1010	0%	-28%	-35%	-39%	-46%	-54%	-57%
exp	0%	-21%	-27%	-30%	-44%	-46%	-45%
m3	0%	-34%	-39%	-39%	-43%	-49%	-45%
misex3	0%	-30%	-40%	-43%	-52%	-52%	-53%
spla	0%	-20%	-23%	-25%	-30%	-35%	-34%
table3	0%	-21%	-31%	-33%	-42%	-45%	-50%
table5	0%	-23%	-33%	-39%	-47%	-52%	-58%
test1	0%	-15%	-20%	-22%	-30%	-36%	-44%
test4	0%	-22%	-27%	-29%	-36%	-42%	-43%
<b>Average</b>	0%	-24%	-31%	-33%	-41%	-45%	-48%



Table 6.6: Area overhead for combinational benchmark circuits.

Circuits	Original	Comparison Espresso	Algorithm 1 - Redundancy Threshold				
			Irredundant	0.05	0.01	0.005	0.01-Duplicate
apex3	2302	4622	4769	4782	5239	5532	5461
apex4	3010	4446	4959	4983	5478	5932	5762
bench1	1487	1867	2030	2062	2374	2622	2639
cps	1728	2183	2264	2371	2660	2829	2844
duke2	632	841	904	934	1046	1150	1073
ex1010	4712	6777	7537	7695	8995	9957	9573
exp	430	567	592	610	728	796	732
m3	358	537	564	564	601	691	605
misex3	1020	1369	1473	1516	1710	1854	1728
spla	555	692	700	710	830	910	859
table3	1171	1445	1579	1597	1804	1966	1924
table5	1333	1644	1783	1873	2138	2391	2301
test1	1189	1492	1589	1624	1903	2091	2124
test4	3083	4171	4535	4628	5287	5776	5577

Circuits	Original	Comparison Espresso	Algorithm 1 - Redundancy Threshold				
			Irredundant	0.05	0.01	0.005	0.01-Duplicate
apex3	0%	101%	107%	108%	128%	140%	137%
apex4	0%	48%	65%	66%	82%	97%	91%
bench1	0%	26%	37%	39%	60%	76%	77%
cps	0%	26%	31%	37%	54%	64%	65%
duke2	0%	33%	43%	48%	66%	82%	70%
ex1010	0%	44%	60%	63%	91%	111%	103%
exp	0%	32%	38%	42%	69%	85%	70%
m3	0%	50%	58%	58%	68%	93%	69%
misex3	0%	34%	44%	49%	68%	82%	69%
spla	0%	25%	26%	28%	50%	64%	55%
table3	0%	23%	35%	36%	54%	68%	64%
table5	0%	23%	34%	41%	60%	79%	73%
test1	0%	25%	34%	37%	60%	76%	79%
test4	0%	35%	47%	50%	71%	87%	81%
<b>Average</b>	0%	38%	47%	50%	70%	86%	79%

Table 6.7: Overall averaged failure rate results for benchmark circuits.

Circuits	Original	Comparison Espresso	Algorithm 1 - Redundancy Threshold				
			Irredundant	0.05	0.01	0.005	0.01-Duplicate
1	0%	-36%	-46%	-49%	-55%	-60%	-62%
5	0%	-30%	-39%	-41%	-49%	-53%	-56%
10	0%	-24%	-31%	-33%	-41%	-45%	-48%
Average	0%	-30%	-38%	-41%	-48%	-52%	-55%

## 6.4 Algorithm 2 Results

In this section, we will apply Algorithm 2 to combinational benchmark circuits resulting after applying Algorithm 1. The aggregated results are reported and discussed in this section. The experiments for Algorithm 2 specified in Section 6.1 are implemented. Results of failure rate for benchmark circuits by injecting 1, 5, and 10 faults are shown in Table 6.8 , Table 6.9 and Table 6.10. In those tables, the amount of failure rate increase after applying Algorithm 2 compared to Algorithm 1 with 0.01 threshold version using duplication is stated and averaged among all circuits.

From these tables, we can draw the following conclusions:

- By injecting 1 fault, the best failure rate increase is 7% for 0.1 threshold using Algorithm 2. This is followed by 11% failure rate increase for 0.3 threshold, then 16% failure rate increase for fx for area experiment.

- By injecting 5 faults, the best failure rate increase is 5% for 0.1 threshold using Algorithm 2. This is followed by 8% failure rate increase for 0.3 threshold, then 13% failure rate increase for fx for area experiment.
- By injecting 10 faults, the best failure rate increase is 6% for 0.1 threshold using Algorithm 2. This is followed by 8% failure rate overhead for 0.3 increase, then 11% failure rate increase for fx for area experiment.

The area for each experiment in each circuit is shown in Table 6.11. The most area reduction on average is 13% for fx for area experiment. This is followed by 11% for 0.3 threshold using algorithm 2 then 10% for 0.1 threshold.

The overall averages of failure rate overhead derived from the previously mentioned tables are shown in Table 6.12. We can say that on average, the best failure rate increase is 6% for 0.1 threshold using Algorithm 2. This is followed by 9% failure rate increase for 0.3 threshold, then 13% failure rate increase for fx-area experiment.

Based on the results in this section, we can draw the following conclusions:

- Using Algorithm 2 with 0.1 threshold, we can get around 10% reduction in area compared to area obtained after applying Algorithm 1. However, the failure rate can increase by 6%.
- Using Algorithm 2, we can get better circuits in terms of reliability than using the original fx algorithm to optimize area. However, original fx gives better circuits in terms of area.

Table 6.8: Failure rate results for combinational benchmark circuits using Algorithm 2  
- Injecting 1 fault.

Circuits	Algorithm 1 - Threshold = 0.01	fx-Area	Algorithm 2 - Threshold	
			0.3	0.1
apex3	0.042	0.066	0.061	0.051
apex4	0.036	0.039	0.038	0.037
bench1	0.051	0.056	0.054	0.052
cps	0.067	0.070	0.069	0.068
duke2	0.063	0.069	0.064	0.063
ex1010	0.028	0.033	0.030	0.030
exp	0.067	0.074	0.072	0.071
m3	0.059	0.070	0.070	0.069
misex3	0.034	0.040	0.039	0.036
spla	0.076	0.084	0.080	0.079
table3	0.038	0.045	0.043	0.040
table5	0.034	0.042	0.039	0.037
test1	0.058	0.060	0.059	0.059
test4	0.056	0.063	0.062	0.061

Circuits	Algorithm 1 - Threshold = 0.01	fx-Area	Algorithm 2 - Threshold	
			0.3	0.1
apex3	0%	57%	46%	23%
apex4	0%	8%	6%	4%
bench1	0%	11%	7%	3%
cps	0%	4%	3%	1%
duke2	0%	10%	2%	1%
ex1010	0%	14%	6%	5%
exp	0%	11%	8%	6%
m3	0%	19%	19%	17%
misex3	0%	16%	13%	6%
spla	0%	11%	5%	3%
table3	0%	19%	13%	6%
table5	0%	21%	13%	8%
test1	0%	3%	1%	1%
test4	0%	13%	11%	10%
<b>Average</b>	0%	16%	11%	7%

Table 6.9: Failure rate results for combinational benchmark circuits using Algorithm 2  
- Injecting 5 faults.

Circuits	Algorithm 1 - Threshold = 0.01	fx-Area	Algorithm 2 - Threshold	
			0.3	0.1
apex3	0.193	0.278	0.254	0.247
apex4	0.169	0.192	0.177	0.172
bench1	0.208	0.223	0.216	0.210
cps	0.281	0.305	0.295	0.290
duke2	0.259	0.298	0.283	0.279
ex1010	0.130	0.153	0.148	0.140
exp	0.294	0.315	0.304	0.299
m3	0.267	0.280	0.280	0.278
misex3	0.160	0.178	0.170	0.164
spla	0.329	0.356	0.344	0.335
table3	0.180	0.193	0.191	0.189
table5	0.169	0.196	0.176	0.175
test1	0.235	0.256	0.249	0.240
test4	0.243	0.263	0.261	0.255

Circuits	Algorithm 1 - Threshold = 0.01	fx-Area	Algorithm 2 - Threshold	
			0.3	0.1
apex3	0%	44%	32%	28%
apex4	0%	14%	5%	2%
bench1	0%	7%	3%	1%
cps	0%	8%	5%	3%
duke2	0%	15%	9%	7%
ex1010	0%	18%	14%	8%
exp	0%	7%	3%	2%
m3	0%	5%	5%	4%
misex3	0%	11%	7%	3%
spla	0%	8%	5%	2%
table3	0%	7%	6%	5%
table5	0%	16%	4%	3%
test1	0%	9%	6%	2%
test4	0%	8%	7%	5%
<b>Average</b>	0%	13%	8%	5%

Table 6.10: Failure rate results for combinational benchmark circuits using Algorithm 2 - Injecting 10 faults.

Circuits	Algorithm 1 - Threshold = 0.01	fx-Area	Algorithm 2 - Threshold	
			0.3	0.1
apex3	0.340	0.470	0.444	0.417
apex4	0.304	0.338	0.331	0.330
bench1	0.367	0.402	0.395	0.391
cps	0.481	0.519	0.502	0.493
duke2	0.442	0.504	0.467	0.456
ex1010	0.247	0.280	0.263	0.260
exp	0.475	0.513	0.510	0.497
m3	0.435	0.460	0.460	0.459
misex3	0.284	0.314	0.311	0.308
spla	0.541	0.573	0.554	0.549
table3	0.327	0.354	0.351	0.340
table5	0.309	0.336	0.312	0.310
test1	0.424	0.441	0.435	0.430
test4	0.418	0.457	0.451	0.445

Circuits	Algorithm 1 - Threshold = 0.01	fx-Area	Algorithm 2 - Threshold	
			0.3	0.1
apex3	0%	38%	31%	23%
apex4	0%	11%	9%	8%
bench1	0%	10%	8%	6%
cps	0%	8%	4%	3%
duke2	0%	14%	6%	3%
ex1010	0%	13%	7%	5%
exp	0%	8%	7%	5%
m3	0%	6%	6%	5%
misex3	0%	11%	10%	8%
spla	0%	6%	2%	1%
table3	0%	8%	7%	4%
table5	0%	9%	1%	0%
test1	0%	4%	2%	1%
test4	0%	9%	8%	6%
<b>Average</b>	0%	11%	8%	6%

Table 6.11: Area reduction for combinational benchmark circuits using Algorithm 2.

Circuits	Redundant-0.01	fx-Area	Algorithm 2 - Threshold	
			0.3	0.1
apex3	5461	3512	3739	3896
apex4	5762	5083	5132	5214
bench1	2639	2429	2436	2471
cps	2844	2591	2649	2682
duke2	1073	912	958	972
ex1010	9573	8262	8341	8489
exp	732	638	663	672
m3	605	538	542	553
misex3	1728	1546	1570	1580
spla	859	757	771	785
table3	1924	1749	1756	1787
table5	2301	2102	2131	2141
test1	2124	1976	1994	2001
test4	5577	4917	4946	5018

Circuits	Redundant-0.01	fx-Area	Algorithm 2 - Threshold	
			0.3	0.1
apex3	0%	-36%	-32%	-29%
apex4	0%	-12%	-11%	-10%
bench1	0%	-8%	-8%	-6%
cps	0%	-9%	-7%	-6%
duke2	0%	-15%	-11%	-9%
ex1010	0%	-14%	-13%	-11%
exp	0%	-13%	-9%	-8%
m3	0%	-11%	-10%	-9%
misex3	0%	-11%	-9%	-9%
spla	0%	-12%	-10%	-9%
table3	0%	-9%	-9%	-7%
table5	0%	-9%	-7%	-7%
test1	0%	-7%	-6%	-6%
test4	0%	-12%	-11%	-10%
<b>Average</b>	0%	-13%	-11%	-10%

Table 6.12: Overall averaged failure rate overhead for benchmark circuits using Algorithm 2.

Injected Faults	Algorithm1-0.01	fx-Area	Algorithm 2 - Threshold	
			0.3	0.1
1	0%	16%	11%	7%
5	0%	13%	8%	5%
10	0%	11%	8%	6%
<b>Average</b>	0%	13%	9%	6%

## 6.5 Overall Results Using Algorithm 1 and Algorithm 2

In this section, the overall results of our technique including applying Algorithm 1 and Algorithm 2 compared to the original circuits is presented. The results are in terms of failure rate and area overhead. Two versions are used:

- Original: In this version, the original benchmark circuit is used.
- Proposed: In this version, the resulting circuit after applying Algorithm 1 followed by Algorithm 2 on the original circuit is used. A threshold of 0.01 with duplication is used for Algorithm 1. Algorithm 2 is applied using a threshold of 0.1.

Table 6.13 shows the failure rate results of the proposed version compared to the original version for all used benchmarks. In this table, fixed number of faults is injected in both versions: 1, 5 and 10 faults. On average, results show that the proposed version reduces failure rate by 59% if one fault is injected, 53% if 5 faults are injected and 45% if 10 faults are injected. Table 6.15 shows the area overhead of the proposed version



compared to the original one. It shows that proposed version adds an area overhead of 61% on average compared to original version.

In the previous results, for each circuit, we are comparing the original circuit with the proposed circuit by finding the failure rate for a fixed number of faults. However, another comparison that can be made is by making the number of injected faults proportional to circuit area. For instance, if we inject one fault in the original circuit, and the area of the new proposed circuit is twice the original, then we inject 2 faults in the new circuit. According to area overhead shown in Table 6.15, we can derive the number of faults that should be injected for synthesized circuits if  $k$  faults are injected in the original circuit. Table 6.14 shows the failure rate results if a fixed number of faults are injected in the original version: 1, 5 and 10 faults. The number of faults injected in the proposed version is proportional to its area compared to original area. On average, results show that the proposed version reduces failure rate by 24% if one fault is injected in the original version, 31% if 5 faults are injected and 24% if 10 faults are injected.

Table 6.13: Overall results of failure rate for benchmark circuits using fixed number of faults 1, 5 and 10.

Circuits	1 Fault		5 Faults		10 Faults	
	original	proposed	original	proposed	original	proposed
apex3	0.119	0.051	0.467	0.247	0.713	0.417
apex4	0.093	0.037	0.379	0.172	0.620	0.330
bench1	0.131	0.052	0.503	0.210	0.739	0.391
cps	0.166	0.068	0.595	0.290	0.841	0.493
duke2	0.152	0.063	0.567	0.279	0.793	0.456
ex1010	0.083	0.030	0.357	0.140	0.572	0.260
exp	0.185	0.071	0.632	0.299	0.861	0.497
m3	0.152	0.069	0.540	0.278	0.792	0.459
misex3	0.087	0.036	0.378	0.164	0.600	0.308
spla	0.163	0.079	0.594	0.335	0.820	0.549
table3	0.108	0.040	0.430	0.189	0.660	0.340
table5	0.123	0.037	0.476	0.175	0.727	0.310
test1	0.136	0.059	0.513	0.240	0.762	0.430
test4	0.130	0.061	0.494	0.255	0.735	0.445

Circuits	1 Fault		5 Faults		10 Faults	
	original	proposed	original	proposed	original	proposed
apex3	0%	-57%	0%	-47%	0%	-42%
apex4	0%	-60%	0%	-55%	0%	-47%
bench1	0%	-60%	0%	-58%	0%	-47%
cps	0%	-59%	0%	-51%	0%	-41%
duke2	0%	-59%	0%	-51%	0%	-43%
ex1010	0%	-64%	0%	-61%	0%	-55%
exp	0%	-62%	0%	-53%	0%	-42%
m3	0%	-55%	0%	-49%	0%	-42%
misex3	0%	-59%	0%	-57%	0%	-49%
spla	0%	-52%	0%	-44%	0%	-33%
table3	0%	-63%	0%	-56%	0%	-49%
table5	0%	-70%	0%	-63%	0%	-57%
test1	0%	-57%	0%	-53%	0%	-44%
test4	0%	-53%	0%	-48%	0%	-39%
Average	0%	-59%	0%	-53%	0%	-45%

Table 6.14: Overall results of failure rate for benchmark circuits using fixed number of faults 1, 5 and 10 at the original version.

Circuits	1 Fault at original			5 Faults at original			10 Faults at original		
	Fail Rate original	proposed		Fail Rate original	proposed		Fail Rate original	proposed	
		Faults	Fail Rate		Faults	Fail Rate		Faults	Fail Rate
apex3	0.119	2	0.1059	0.467	8	0.3655	0.713	17	0.5981
apex4	0.093	2	0.0795	0.379	9	0.2705	0.620	17	0.4801
bench1	0.131	2	0.0897	0.503	8	0.3300	0.739	17	0.5555
cps	0.166	2	0.1270	0.595	8	0.4168	0.8407	16	0.6832
duke2	0.152	2	0.1231	0.567	8	0.4037	0.7934	15	0.6107
ex1010	0.083	2	0.0598	0.357	9	0.2332	0.572	18	0.4204
exp	0.185	2	0.1363	0.632	8	0.4312	0.861	16	0.6660
m3	0.152	2	0.1313	0.540	8	0.3920	0.792	15	0.5778
misex3	0.087	2	0.0728	0.378	8	0.2524	0.600	15	0.4139
spla	0.163	1	0.0847	0.594	7	0.4366	0.8199	14	0.6645
table3	0.108	2	0.0782	0.430	8	0.2839	0.6596	15	0.4632
table5	0.123	2	0.0746	0.476	8	0.2590	0.7274	16	0.4531
test1	0.136	2	0.1072	0.513	8	0.3660	0.762	17	0.6208
test4	0.130	2	0.1148	0.494	8	0.3694	0.735	16	0.6054

Circuits	1 Fault at original			5 Faults at original			10 Faults at original		
	Fail Rate original	proposed		Fail Rate original	proposed		Fail Rate original	proposed	
		Faults	Fail Rate		Faults	Fail Rate		Faults	Fail Rate
apex3	0%	2	-11%	0%	8	-22%	0%	17	-16%
apex4	0%	2	-15%	0%	9	-29%	0%	17	-23%
bench1	0%	2	-31%	0%	8	-34%	0%	17	-25%
cps	0%	2	-24%	0%	8	-30%	0%	16	-19%
duke2	0%	2	-19%	0%	8	-29%	0%	15	-23%
ex1010	0%	2	-28%	0%	9	-35%	0%	18	-27%
exp	0%	2	-26%	0%	8	-32%	0%	16	-23%
m3	0%	2	-14%	0%	8	-27%	0%	15	-27%
misex3	0%	2	-16%	0%	8	-33%	0%	15	-31%
spla	0%	1	-48%	0%	7	-27%	0%	14	-19%
table3	0%	2	-28%	0%	8	-34%	0%	15	-30%
table5	0%	2	-39%	0%	8	-46%	0%	16	-38%
test1	0%	2	-21%	0%	8	-29%	0%	17	-19%
test4	0%	2	-12%	0%	8	-25%	0%	16	-18%
Average			-24%			-31%			-24%

Table 6.15: Area overhead for combinational benchmark circuits using proposed technique compared to original circuits.

Circuits	Original	Proposed	Overhead
<b>apex3</b>	2302	3896	69%
<b>apex4</b>	3010	5214	73%
<b>bench1</b>	1487	2471	66%
<b>cps</b>	1728	2682	55%
<b>duke2</b>	632	972	54%
<b>ex1010</b>	4712	8489	80%
<b>exp</b>	430	672	56%
<b>m3</b>	358	553	54%
<b>misex3</b>	1020	1580	55%
<b>spla</b>	555	785	41%
<b>table3</b>	1171	1787	53%
<b>table5</b>	1333	2141	61%
<b>test1</b>	1189	2001	68%
<b>test4</b>	3083	5018	63%
		<b>Average</b>	61%

## 6.6 Comparison Results

In this section, a comparison between Algorithm1 results and the reliability driven don't care assignment technique described in Chapter 2 is reported. Four versions are used to compare with reliability driven don't care assignment technique[37]:

- Algorithm1-Irredundant: No duplication is used.
- Algorithm1-0.01: In this version, Algorithm1 is applied on extracted sub-circuit using a threshold value of 0.01. No duplication is used.
- Algorithm1-0.005: In this version, Algorithm1 is applied on extracted sub-circuit using a threshold value of 0.005. No duplication is used.

- Ranking-based Don't Care: In this version, the ranking-based DC assignment algorithm proposed in [37] is used. This algorithm is applied on each extracted sub-circuit using a fraction value of 0.6.

To perform the comparison, a fixed number of faults (10 faults) are injected in circuits of Algorithm1-Irredundant version. For the other versions, the number of injected faults is proportional to the area in each version if 10 faults are injected in the irredundant version. Failure rate results of the previous experiments are shown in Table 6.16. The results show that the ranking-based DC assignment technique gives higher failure rates and higher area overhead compared to Algorithm 1. On average, it increases the failure rate by 14% compared to Algorithm 1 implementation with 0.05-threshold, 14% compared to 0.01 threshold and 7% compared to the Irredundant version.

The area increase is reported in Table 6.17. It's shown that on average, a 5% area increase is added when using ranking-based DC assignment algorithm compared to Algorithm 1 with 0.05-threshold, 15% compared to 0.01 threshold and 33% compared to the Irredundant version. Algorithm 1 shows better failure rate results compared to ranking-based DC assignment algorithm results with less area overhead. This is due to carefully selecting cubes to cover minterms according to probability of minterms instead of blindly assigning don't cares to either on-set or off-set.

Table 6.16: Failure rate results for Ranking-based Don't Care comparison.

Circuits	Algorithm1-Irredundant		Algorithm1-0.01		Algorithm1-0.005		Ranking-based DC		Failure Rate Increase		
	Faults	Failure Rate	Faults	Failure Rate	Faults	Failure Rate	Faults	Failure Rate	Irred.	Thr-0.01	Thr-0.005
apex3	10	0.401	11	0.395	12	0.398	10	0.417	4%	5%	5%
apex4	10	0.406	11	0.377	12	0.379	15	0.448	10%	18%	18%
bench1	10	0.559	12	0.540	13	0.539	13	0.584	4%	8%	8%
cps	10	0.647	12	0.625	12	0.599	11	0.673	4%	12%	12%
duke2	10	0.536	12	0.526	13	0.527	10	0.582	9%	10%	10%
ex1010	10	0.373	12	0.341	13	0.341	13	0.380	2%	12%	12%
exp	10	0.625	12	0.544	13	0.552	13	0.653	5%	18%	18%
m3	10	0.484	11	0.468	12	0.452	23	0.522	8%	15%	15%
misex3	10	0.361	12	0.340	13	0.341	16	0.445	23%	30%	30%
spla	10	0.630	12	0.634	13	0.639	11	0.651	3%	2%	2%
table3	10	0.455	11	0.413	12	0.409	11	0.483	6%	18%	18%
table5	10	0.486	12	0.451	13	0.432	11	0.546	12%	26%	26%
test1	10	0.610	12	0.581	13	0.583	13	0.626	3%	7%	7%
test4	10	0.538	12	0.540	13	0.503	15	0.595	11%	18%	18%
<b>Average</b>									7%	14%	14%

Table 6.17: Area overhead of Ranking-based DC compared to Algorithm 1 versions.

Circuits	Algorithm1-Irredundant	Algorithm1-0.01	Algorithm1-0.005	Ranking-based DC	Area Increase		
					Irred.	Thr-0.01	Thr-0.005
apex3	4769	5239	5532	4875	2%	-7%	-12%
apex4	4959	5478	5932	7234	46%	32%	22%
bench1	2030	2374	2622	2623	29%	10%	0%
cps	2264	2660	2829	2448	8%	-8%	-13%
duke2	904	1046	1150	947	5%	-9%	-18%
ex1010	7537	8995	9957	10061	33%	12%	1%
exp	592	728	796	769	30%	6%	-3%
m3	564	605	691	1317	134%	118%	91%
misex3	1473	1710	1854	2355	60%	38%	27%
spla	700	830	910	796	14%	-4%	-13%
table3	1579	1804	1966	1802	14%	0%	-8%
table5	1783	2138	2391	1981	11%	-7%	-17%
test1	1589	1903	2091	2101	32%	10%	0%
test4	4535	5287	5776	6710	48%	27%	16%
<b>Average</b>					33%	15%	5%

## 6.7 Proposed Technique with TMR

In this section, we integrate our proposed method with TMR technique. That is, instead of using the original circuit in each module in TMR, we use the circuit after applying our technique using Algorithm 1 and 2. Two versions are used in this section:

- TMR-original: In this version, the original circuit is triplicated.
- TMR-proposed: In this version, the circuit resulting after applying Algorithm1 with threshold 0.01(with duplication) and Algorithm2 with 0.1 threshold is triplicated.

Fixed number of faults (10 faults) is injected in the TMR-original version. The number of faults injected in the TMR-proposed version is proportional to its area compared to TMR-original version. Table 6.18 shows the failure results of the previous two experiments. It shows that TMR-proposed failure rates show a significant reduction compared to TMR-original. This reduction is 27% on average. This is due to the decrease in the probability for two modules to fail. The area overhead is reported in Table 6.19. It's shown that on average, a 57% area overhead is added when using the proposed circuit instead of the original circuit in the TMR structure.

Table 6.18: Failure rate results for TMR experiments.

Circuits	TMR -Original		TMR - Algo1 -Algo2		Failure Rate Reduction
	Injected Faults	Failure Rate	Injected Faults	Failure Rate	
apex3	10	0.1365	17	0.1171	-14%
apex4	10	0.0604	17	0.0443	-27%
bench1	10	0.1196	16	0.0622	-48%
cps	10	0.2301	15	0.1994	-13%
duke2	10	0.212	15	0.1852	-13%
ex1010	10	0.0557	18	0.0255	-54%
exp	10	0.2465	15	0.181	-27%
m3	10	0.2275	15	0.2043	-10%
misex3	10	0.0967	15	0.0818	-15%
spla	10	0.2956	13	0.2556	-14%
table3	10	0.1061	15	0.0731	-31%
table5	10	0.1364	16	0.0711	-48%
test1	10	0.1161	17	0.0773	-33%
test4	10	0.1161	16	0.0773	-33%
				<b>Average</b>	-27%

Table 6.19: Area overhead for TMR experiments.

Circuits	TMR -Original	TMR - Algo1 -Algo2	Overhead
apex3	7191	11929	66%
apex4	9174	15710	71%
bench1	4551	7476	64%
cps	5919	8626	46%
duke2	2111	3106	47%
ex1010	14181	25534	80%
exp	1383	2116	53%
m3	1165	1759	51%
misex3	3156	4771	51%
spla	1995	2640	32%
table3	3599	5421	51%
table5	4080	6445	58%
test1	3635	6088	67%
test4	9488	15228	60%
		<b>Average</b>	57%



## Chapter 7

# CONCLUSION AND FUTURE

## WORK

Recently, systems became more subjected to higher manufacturing defects and higher susceptibility to soft errors due to the exponential decrease in device feature size. Currently, soft errors induced by ion particles are no longer limited to specific field such as aerospace applications. This raises the challenge to come up with techniques to tackle transient or soft error effects in both combinational and sequential circuits in general. In this work, we have analyzed, modeled and designed combinational circuits to increase its immunity to radiation induced transient faults or soft errors.

We have introduced a novel idea to increase combinational circuit reliability and hence fault tolerance. This idea is based on extracting sub-circuits from the original multi-level circuit and re-synthesizing each extracted sub-circuit to increase fault masking. After that, the re-synthesized sub-circuits are merged back to the original circuit. Therefore, the overall reliability of the original circuit will be enhanced as well. We have presented a two-level synthesis scheme to maximize soft error masking that is applied on each extracted circuit, presented as Algorithm 1. This scheme provides a heuristic that first finds the best irredundant set of cubes to cover an extracted sub-circuit

minterms such that fault masking for single fault is maximized especially for minterms with high probability of occurrence. Then, an extra number of cubes can be added as redundant cubes to the cover such that they have a significant effect on maximizing error masking.

Algorithm 1 experimental results show that the best failure rate reduction compared to the original circuit is found to be 55% on average using 0.01 threshold with duplication, 52% reduction for 0.005 threshold, 48% reduction for 0.01 threshold without duplication, 41% reduction for 0.05 threshold and 38% for the irredundant version. The area overhead is on average 79%, 86%, 70%, 50% and 47% respectively. Hence, a threshold of value 0.01 with duplication is recommended. This value provides the best tradeoff between area overhead and improved fault tolerance.

A technique based on modification of the fast extraction algorithm, presented as Algorithm 2, has been proposed to enhance area overhead to optimized circuits obtained by Algorithm 1. Algorithm 2 results show that we can get around 10% reduction in area compared to area obtained after applying Algorithm 1, with a failure rate increase by 6%.

The final results after applying Algorithm 1 followed by Algorithm 2 are reported. Algorithm 1 is used with the recommended threshold of 0.01 with duplication. Algorithm 2 is used with 0.1 threshold. If a fixed number of faults are injected in both the original circuit and the new re-synthesized circuit, an average failure rate reduction of 52% is obtained compared to original circuit. However, if the number of injected faults in the new re-synthesized circuit is proportional to area compared to original area, an average

failure rate reduction of 26% is obtained compared to the original circuit. An average area overhead of 61% is added compared to the original circuit.

## 7.1 Summary of the Contributions

The contributions of this thesis can be summarized as follows:

- Implemented a tool for extracting sub-circuits from an original multi-level circuit, finding inputs don't cares, and the probability of care minterms for each extracted sub-circuit.
- Implemented a tool for computing soft error reliability for combinational circuits based on Monte Carlo simulation [10]. The objective of this tool is to find the failure rate of a combinational circuit as more faults are observed in the circuit. The Monte Carlo based simulation tool has been developed using C# [52] in Windows system to assess the soft error reliability of the resulting synthesized circuits.
- Developed and implemented an algorithm to enhance reliability of combinational circuits based on enhancing reliability of each individual circuit. A two-level synthesis heuristic has been proposed and applied on each extracted sub-circuit that attempts to find the best cover that maximizes fault masking while keeping area overhead minimum.
- Implemented a multi-level tool based on the fast extraction algorithm that reduces the area overhead resulting after applying the two-level tool.
- Evaluated the proposed approaches in terms of failure rate and area overhead.

## 7.2 Future Work

In our work, only the logic masking property was considered to maximize fault masking. As a future work, the latching window masking property can be taken into account as well. This can help in reducing the area overhead resulting from adding more redundant cubes. In other words, if a fault in a certain sub-circuit can be masked according to latching window property, then no need to add redundant cubes for that sub-circuit.

An enhancement for the reliability driven don't care assignment technique[37] can be done as a future work too. By getting benefit from probabilities of care minterms, don't cares can be assigned to either on or off based on the probabilities of neighboring minterms. If the neighboring on-minterms have higher probability of occurrence, a don't care is assigned to on-set. Otherwise it's assigned to off-set. If the probability of occurrence for on and off neighboring minterms are close to each other, a don't care isn't assigned to any phase.

## REFERENCES

- [1] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, April 1965.
- [2] M. Butts, *et al.*, "Molecular electronics: devices, systems and tools for gigagate, gigabit chips," in *Computer Aided Design, 2002. ICCAD 2002. IEEE/ACM International Conference 2002*, pp. 433-440.
- [3] A. Bachtold, *et al.*, "Logic circuits with carbon nanotube transistors," *Science*, vol. 294, pp. 1317-1320, November 2001.
- [4] Y. Cui and C. M. Lieber, "Functional nanoscale electronic devices assembled using silicon nanowire building blocks," *Science*, vol. 291, pp. 851-853, February 2001.
- [5] Y. Huang, *et al.*, "Logic gates and computation from assembled nanowire building blocks," *Science*, vol. 294, pp. 1313-1317, November 2001.
- [6] P. D. Tougaw and C. S. Lent, "Logical devices implemented using quantum cellular automata," *J. Applied physics*, vol. 75, pp. 1818-1825, 1994.
- [7] J. Han, "Fault-tolerant architectures for nanoelectronic and quantum devices," Ph.D. dissertation, Dept. Comp. Eng. , Delft University of Technology, 2004.
- [8] D. G. Mavis and P. H. Eaton, "Temporally redundant latch for preventing single event disruptions in sequential integrated circuits," Mission Research Corp., Tech. Rep. P8111, Sep. 1998.
- [9] B. S. Gill, "Design and analysis methodologies to reduce soft errors in nanometer VLSI circuits," Ph.D. dissertation, Dept. Elect. Eng., Case Western Reserve University, 2006.

- [10] M. H. Kalos and P. A. Whitlock, *Monte carlo methods*: Wiley-VCH, 2008.
- [11] M. L. Bushnell and V. D. Agrawal, *Essentials of electronic testing for digital, memory, and mixed-signal VLSI circuits* vol. 17: Springer Netherlands, 2000.
- [12] C. E. Ebeling, *An introduction to reliability and maintainability engineering*: Waveland Press, Inc, Illinois, 1997.
- [13] T. P. Ma and P. V. Dressendorfer, *Ionizing radiation effects in MOS devices and circuits*: Wiley-Interscience, 1989.
- [14] S. Mitra, *et al.*, "Robust system design with built-in soft-error resilience," *Computer*, vol. 38, pp. 43-52, 2005.
- [15] N. Miskov-Zivanov and D. Marculescu, "Soft error rate analysis for sequential circuits," in *DATE '07: Design, Automation & Test in Europe Conference & Exhibition*, 2007, pp. 1-6.
- [16] D. P. Siewiorek, *et al.*, *Reliable computer systems: design and evaluation* vol. 2: Digital Press, 1992.
- [17] A. Avižienis, "Design of fault-tolerant computers," in *Joint Computer Conf. of AFIPS*, 1967, pp. 733-743.
- [18] P. K. Lala, *Self-checking and fault-tolerant digital design*: Morgan Kaufmann Pub, 2001.
- [19] J. R. Heath, *et al.*, "A defect-tolerant computer architecture: Opportunities for nanotechnology," *Science*, vol. 280, pp. 1716-1721, 1998.
- [20] C. He, *et al.*, "A reconfiguration-based defect-tolerant design paradigm for nanotechnologies," *Design & Test of Computers, IEEE*, vol. 22, pp. 316-326, 2005.

- [21] M. Mishra and S. Goldstein, "Defect tolerance at the end of the roadmap," *Nano, quantum and molecular computing*, pp. 73-108, 2004.
- [22] M. B. Tahoori, "Application-independent defect tolerance of reconfigurable nanoarchitectures," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 2, pp. 197-218, 2006.
- [23] T. Hogg and G. S. Snider, "Defect-tolerant adder circuits with nanoscale crossbars," *Nanotechnology, IEEE Transactions on*, vol. 5, pp. 97-100, 2006.
- [24] M. Gössel and S. Graf, *Error detection circuits*: McGraw-Hill, 1993.
- [25] M. Nicolaidis and Y. Zorian, "On-line testing for VLSI—A compendium of approaches," *Journal of Electronic Testing*, vol. 12, pp. 7-20, 1998.
- [26] J. Von Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components," *Automata studies*, vol. 34, pp. 43-98, 1956.
- [27] Y. Qi, *et al.*, "Markov chains and probabilistic computation—a general framework for multiplexed nanoelectronic systems," *Nanotechnology, IEEE Transactions on*, vol. 4, pp. 194-205, 2005.
- [28] J. Han and P. Jonker, "A defect-and fault-tolerant architecture for nanocomputers," *Nanotechnology*, vol. 14, pp. 224-230, 2003.
- [29] A. S. Sadek, *et al.*, "Parallel information and computation with restitution for noise-tolerant nanoscale logic networks," *Nanotechnology*, vol. 15, pp. 192-210, 2004.
- [30] W. H. Pierce and P. A. Jensen, *Failure-tolerant computer design*: Academic Press, 1965.

- [31] J. Tryon, *Quadded logic Redundancy Techniques for Computing Systems*: Spartan Books, 1962.
- [32] P. A. Jensen, "Quadded NOR logic," *Reliability, IEEE Transactions on*, vol. 12, pp. 22-31, 1963.
- [33] K. Mohanram and N. A. Touba, "Partial error masking to reduce soft error failure rate in logic circuits," *Proc. IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, 2003.
- [34] Y. Dotan, *et al.*, "Fault tolerance for nanotechnology devices at the bit and module levels with history index of correct computation," *Computers & Digital Techniques, IET*, vol. 5, pp. 221-230, 2011.
- [35] A. H. El-Maleh, *et al.*, "Defect-tolerant n2 transistor structure for reliable nanoelectronic designs," *Computers & Digital Techniques, IET*, vol. 3, pp. 570-580, 2009.
- [36] S. Krishnaswamy, *et al.*, "Enhancing design robustness with reliability-aware resynthesis and logic simulation," in *Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on*, Nov. 2007, pp. 149-154.
- [37] A. Zukoski, *et al.*, "Reliability-driven don't care assignment for logic synthesis," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, March 2011, pp. 1-6.
- [38] K. C. Wu and D. Marculescu, "Soft error rate reduction using redundancy addition and removal," in *Design Automation Conference, 2008. ASPDAC 2008. Asia and South Pacific* March 2008, pp. 559-564.



- [39] Q. Zhou and K. Mohanram, "Gate sizing to radiation harden combinational logic," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 25, pp. 155-166, 2006.
- [40] Y. S. Dhillon, *et al.*, "Analysis and optimization of nanometer CMOS circuits for soft-error tolerance," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 14, pp. 514-524, 2006.
- [41] V. Joshi, *et al.*, "Logic SER reduction through flip flop redesign," in *Quality Electronic Design, 2006. ISQED '06. 7th International Symposium* March 2006.
- [42] R. R. Rao, *et al.*, "Soft error reduction in combinational logic using gate resizing and flipflop selection," in *Computer-Aided Design, 2006. ICCAD '06. IEEE/ACM International Conference*, Nov. 2006, pp. 502-509.
- [43] D. B. Limbrick, *et al.*, "Impact of logic synthesis on soft error vulnerability using a 90-nm bulk CMOS digital cell library," in *Southeastcon, 2011 Proceedings of IEEE* pp. 430-434.
- [44] D. Shin and S. K. Gupta, "A new circuit simplification method for error tolerant applications," in *Design, Automation & Test in Europe Conference & Exhibition (DATE) 2011*, pp. 1-6.
- [45] E. M. Sentovich, *et al.*, "SIS: A system for sequential circuit synthesis," EECS Department, University of California, Berkeley, Tech. Rep.,1992.
- [46] N. Saluja and S. P. Khatri, "A robust algorithm for approximate compatible observability don't care (CODC) computation," in *Design Automation Conference, 2004. Proceedings. 41st* 2004, pp. 422-427.

- [47] A. Mishchenko and R. K. Brayton, "SAT-based complete don't-care computation for network optimization," in *Design, Automation and Test in Europe, 2005. Proceedings 2005*, pp. 412-417.
- [48] S. C. Chang, *et al.*, "An efficient algorithm for local don't care sets calculation," in *Design Automation, 1995. DAC '95. 32nd Conference on 1995*, pp. 663-667.
- [49] M. Morris and C. R. Kime, *Logic and computer design fundamentals*: Prentice Hall, 2000.
- [50] J. Rajski and J. Vasudevamurthy, "The testability-preserving concurrent decomposition and factorization of Boolean expressions," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 11, pp. 778-793, 1992.
- [51] J. Han, *et al.*, "Toward hardware-redundant, fault-tolerant logic for nanoelectronics," *Design & Test of Computers, IEEE*, vol. 22, pp. 328-339, 2005.
- [52] H. Deitel, *et al.*, *C#: How to Program*: Prentice Hall, 2002.

# VITA

- General Information:

Name: Khaled Abdel-Karim Daud.  
Place of Birth: Nablus, Palestine.  
Gender: Male.

- Education & Qualification:

- Bachelor of Engineering in Computer Systems from Palestine Polytechnic University, Hebron, Palestine in June 2008.
- Master of Science in Computer Engineering from King Fahd University for Petroleum & minerals (KFUPM), Dhahran, Saudi Arabia in January 2012.

- Work Experience

- Research Assistant in King Fahd University for Petroleum & minerals (KFUPM), Dhahran, Saudi Arabia, from November 2009 to March 2012.
- Computer Laboratory Supervisor in Palestine Polytechnic University, Hebron, Palestine, from September 2008 to September 2009.

- Address and Communication

- Present Address: KFUPM, Dhahran 31261, Saudi Arabia.
- Email: k\_daud@yahoo.com
- Mobile: ++966-596854023