FAULT TOLERANCE TECHNIQUES FOR SEQUENTIAL CIRCUITS: A DESIGN LEVEL APPROACH

by

AYED SAAD AL-QAHTANI

A Thesis Presented to the
DEANSHIP OF GRADUATE STUDIES

In Partial Fulfillment of the Requirements
for the degree

MASTER OF SCIENCE

IN

COMPUTER ENGINEERING

KING FAHD UNIVERSITY
OF PETROLEUM & MINERALS

Dhahran, Saudi Arabia

JUNE 2010

FAULT TOLERANCE TECHNIQUES FOR SEQUENTIAL CIRCUITS: A DESIGN LEVEL APPROACH

BY

AYED SAAD AL-QAHTANI

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

JUNE 2010

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **AYED SAAD AL-QAHTANI** under the direction of his thesis adviser 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 H. El-Maleh (Adviser)

Dr. Sadiq M. Sait (Member)

Dr. Alaaeldin A. M. Amin (Member)

Dr. Basem Al-Madani
Department Chairman

Dr. Salam A. Zummo
Dean of Graduate Studies

13/7/10
Date

This thesis is sincerely dedicated to my loving parents and my siblings

ACKNOWLEDGMENTS

In the name of Allah, Most Gracious, Most Merciful

All praise be to Allah (The One and The Only Creator of everything) for His limitless blessings. May Allah bestow peace and His choicest blessings on the last prophet, Muhammad (Peace Be Upon Him), his family (May Allah be pleased with them), his companions (May Allah be pleased with them) and his followers.

I am extremely grateful to **Almighty, Allah**, who bestowed me the understanding, perseverance and critical thinking to make this accomplishment possible.

I would like to express my appreciation and thanks to my thesis chairman and adviser, **Dr. Aiman El-Maleh**, for his incessant guidance through out thesis completion process. He has been available whenever I needed his assistance. I really appreciate his help in developing ideas as well as his positive comments for improving and bringing out the utmost optimum consequences out of my endeavors. Without his guidance and support, I would have never been able to organize and complete the thesis tasks as optimal and within time constraints.

I would like to express my deep appreciation to **King Fahd University of Petroleum & Minerals** for its support and assistance. Also, I would like to express my thanks and appreciation to thesis committee members **Dr. Sadiq Sait**

and **Dr. Alaaeldin Amin** for their creative comments and guidance through out the thesis completion.

Last but not least, I am highly indebted to all my family for their love and patience. Without their support I would have never been able to finish the thesis. In addition, I express my greatest regards to my friends and colleagues for their support throughout my stay at KFUPM. They have been a steady source for help and encouragement.

Ayed

April 2010

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

THESIS ABSTRACT

NAME: Ayed Saad Al-Qahtani

TITLE OF STUDY: Fault Tolerance Techniques for Sequential Circuits: a Design Level Approach

MAJOR FIELD: Computer Engineering

DATE OF DEGREE: June 2010

With technology advancement (90 nm, 65 nm, 35 nm and even smaller), 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 faults effects in both combinational and sequential circuits in general. This work is directed to analyze, model and design sequential circuits at the design level, namely finite state machine (FSM), to increase its tolerance to radiation induced transient faults. A technique based on adding redundant equivalent states to protect few states with high probability of occurrence is proposed. The added states guarantee that all single bit faults occurring in the state vari-

ables or in their combinational logic of highly occurring states are tolerated. The proposed technique has minimal area overhead because only few number of states are chosen for protection. In addition, state assignment is explored and found to have a minimal impact on soft error tolerance of sequential circuits. Hence, another technique is proposed to enhance reliability to a sequential circuit that has a specific state assignment optimizing a certain criteria such as area. Experimental results on ISCAS89 sequential benchmark circuits show that failure rate reduction of 62% up to 83% is achieved compared to the original circuits. The number of highly probable protected states is 9% up to 40% of the total number of states that yield 25% up to 90% state probability coverage. The area cost is found to be about 1.80 up to 4 times the original circuits area. Moreover, starting from a state assignment which optimizes a sequential circuit in terms of area, similar failure rate reduction is achieved and the resulting area overhead is kept minimal.

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

ARABIC ABSTRACT

ملخص الرسالة

الإسم : عايض بن سعد بن عايض القحطاني.
عنوان الرسالة: أساليب الوقاية من الأخطاء اللحظية: طريقة مستوى تصميمي.
التخصص: هندسة الحاسب الآلي.
تاريخ التخرج : يونيو ٢٠١٠.

مع تقدم التقنية (90 نانومتر، و65 نانومتر، و35 نانومتر، وحتى أصغر من ذلك)، أصبحت الأنظمة أكثر تعرضاً لأخطاء التصنيع وأكثر قابلية للأخطاء الوقتية وذلك بسبب الصغر الشديد في أحجام الأجهزة. حالياً، لم تعد الأخطاء اللحظية المستحثة بواسطة الذرات الأيونية محدودةً بمجال معين مثل التطبيقات الفضائية. لذلك طرق الحماية من الأخطاء الوقتية أصبحت ذات أهمية كبرى، ليس في التطبيقات الفضائية فحسب، بل حتى في الدوائر التوافقية و الدوائر المتتابعة بشكل عام. هذا البحث موجه لتحليل و تمثيل و تصميم الدوائر المتتابعة في مستواها التصميمي، المسمى بجهاز الحالة المنتهية، وذلك لزيادة وقايتها من الأخطاء اللحظية ذات الطابع المستحث أيونياً. اقترحنا طريقة مبنية على إضافة حالات زائدة مساوية لبعض الحالات ذات احتمالية كبيرة للحدوث. الحالات المضافة تضمن تحمل جميع الأخطاء الأحادية التي تحدث في أحد متغيرات الحالات المُعالجة. الطريقة المقترحة ستكون ذات تأثير ضئيل على مساحة الدوائر الكهربائية بسبب إختيارنا لحماية عدد قليل من الحالات. بالإضافة لذلك، تعيين الرموز للحالات بحثت ووجد أنها ذات تأثير قليل على إمكانية تحمل الأخطاء الوقتية في الدوائر الكهربائيةَ المتتابعة. وبالتالي، اقترحنا طريقة أخرى لتحسين اعتمادية الدوائر المتتابعة التي تحتوي على تعيينات حالية محددة التي تحسن معيار معين مثل المساحة. نتائج التجارب على الدوائر المتتابعة القياسية $ISCAS89$ تبين أنه تحقق تقليل في معدل الخطأ من نسبة 62٪ وحتى 83٪ بالمقارنة مع الدوائر الأصلية. كما أن عدد الحالات المحمية ذات الاحتمالِية الكبيرة هو ما يعادل نسبته 9٪ وحتى 40٪ من المجموع الكلي للحالات المحصلة لما نسبته 25٪ وحتى 90٪ من تغطية احتمالية الحالات. كما أنه وجد أن تكلفةَ المساحة ما يقارب 1.8 وحتى أربعة أضعافِ مساحة الدوائر الأصلية. علاوةً على ذلك، بالابتداء من تعيين حالات معين المحسِن لمساحة دائرة متعاقبة ما، تحقق تقليل في معدل الخطأ مساوي لما ذكر معَ تأثير أقل على المساحة.

الكلمات الرئيسة : التسامح في الأخطاء أو العيوب، الأخطاء اللينة، العيوب اللحظية، الحدث الأحادي القالب، الحدث الأحادي اللحظي ، تقنية النانو ، تصميم الأنظمة القوية، اعتمادية الدوائر، الدوائر المتتابعة، معدل الأخطاء اللحظية، الوحدات الثلاثية المكررة.

CHAPTER 1

INTRODUCTION AND MOTIVATION

Vulnerability of digital systems grows in direct proportion to the Moores law [1]. Continuous improvements in CMOS technology entering the nanometer scale has resulted into quantum mechanical effects creating many technological challenges for further scaling of CMOS devices. This has led to the exploration of new technologies for circuit design. Nanotechnology-based fabrication is expected to offer the extra density and potential performance to take electronic circuits the next step. It is estimated that molecular electronics can achieve very high densities (10^{12} devices per cm^2) and operate at very high frequencies (of the order of THz) [2]. Several successful nano-scale electronic devices have been demonstrated by researchers, some of the most promising being carbon nano-tubes (CNT) [3], silicon nano-wires (NW) [4, 5], and quantum dot cells [6]. Nano-scale devices are limited by higher defect rates and increased susceptibility to soft errors. This is

due to the inherent imprecision and randomness in the bottom-up manufacturing process which results in a large number of defective devices during the fabrication process. In addition, the reduced noise tolerance of these devices is responsible for inducing device malfunctions by external influences like EMI, thermal perturbations and cosmic radiations. Therefore, permanent defects may emerge during manufacturing process and transient errors can happen during the operation rendering nano-electronic connections, wires and devices effectively unusable.

Defects or Faults can be classified as either permanent faults or transient faults. The Reliability of a circuit is affected by the amount of both types of faults a circuit can not tolerate. It can be defined as immunity to failures which makes the circuit functions properly despite the existence of those faults. Handling permanent hard faults is much easier than transient soft errors since permanent errors always reside in the circuit the whole time whether the circuit is combinational or sequential. Immunity from transient faults is very crucial especially in sequential circuits since a single fault at a short amount of time can affect the current machine state and can cause the system to end up working in the wrong trace of states.

In order to address issues of unreliability in nano-electronics and to ensure reliable system design and operation, defect tolerant design techniques need to be devised and applied for emerging nano-electronic devices. Enhancing the fault tolerance of circuits against permanent and transient types of faults is required to enhance circuit reliability. In this thesis, we address the fault tolerance of transient faults, known as soft errors. In the following sections, we will introduce

sequential circuit model as well as the nature of transient faults. After that, a glance will be given about this work motivations, objectives and contributions. Then we will finish by thesis organization.

1.1 Sequential Circuits and Finite State Machine

Combinational logic and storage elements, namely flip-flops, are interconnected to form sequential circuits. In other words, sequential circuits are basically circuits which memorize their state. The sequential circuit receives binary information as inputs; together with present state stored by the flip-flops determines the binary value of the next state. There are two types of sequential circuits and their classification depends on the times at which their inputs are observed and their internal state changes. The behavior of *synchronous sequential circuits* can be defined from the knowledge of its signals at discrete instants of times controlled by clock signal. On the other hand, *asynchronous sequential circuit* depends upon the inputs at any instance of time. In this thesis, we are only interested in synchronous sequential circuits. So it is expected that sequential circuits in the following context meant to be synchronous and controlled by a clock. Sequential circuits in which the output depends on the inputs as well as the current states are called *Mealy* model circuits. Otherwise, if the outputs depend only on the current states then they are called *Moore* model circuits. Both of their behavior can be

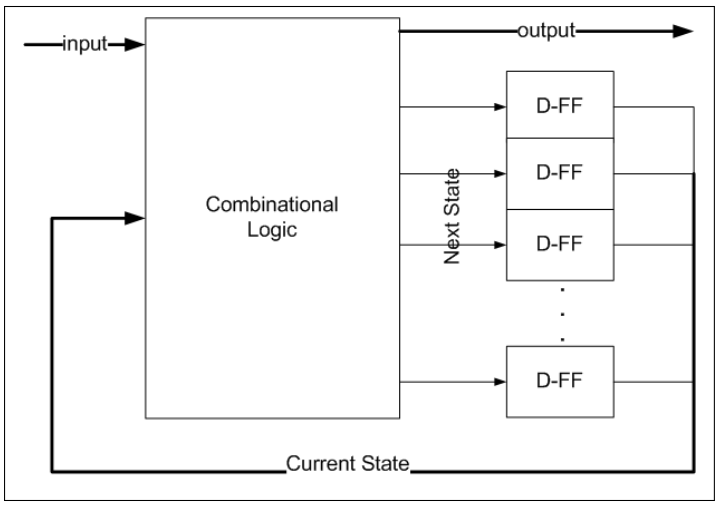


Figure 1.1: Sequential circuit block diagram.

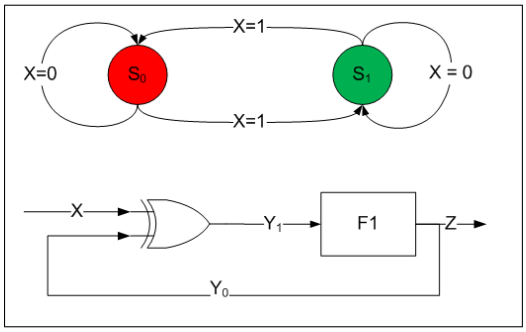


Figure 1.2: A bit flipper.

represented using state table or finite state machine (FSM). The general block diagram for a sequential circuit is shown in Figure 1.1. In Figure 1.1, the output is based on current states and external inputs (mealy machine). In Figure 1.2 we show a bit flipper example. Bit flipper FSM is composed of 2 states, S1 (output =1) and S0 (output = 0). At any specific time the machine is in either S1 or S0, according to the applied input, X. Thus, we need only one flip flop namely F1 to manufacture this simple system. The block diagram for the corresponding FSM is also shown in Figure 1.2. [7–9].

1.2 Soft Errors in Nano-scale Sequential Circuits

Temporal transient faults (soft errors) can hit either in the combinational logic or flip flops of a sequential circuit block diagram as it is depicted in Figure 1.1. If it happens in the combinational logic, it will result in *Single Event Transient* (SET). On the other hand, if it happens in the memory cell itself, it will result in a *Single Event Upset* (SEU). Both of SET and SEU cause a major implication in sequential circuit and should receive a proper treatment.

Transient faults (SET / SEU) mainly are caused by ions movement through the materials of ICs. To illustrate how transient soft errors can change the state of a transistor from OFF to ON or vice versa, we assume the NMOS transistor shown in Figure 1.3 (a) ON. The voltage applied at the gate attracts electrons in the (p-) area, in the same way the holes are attracted by the (p+) substrate. Thus, a channel is generated with a certain amount of carrier electrons that can conduct the two highly doped (n+) regions given that enough voltage is applied to the drain. In normal operation, a current will flow from the drain to the source that make the transistor ON. Suppose an energized ion strike through the drain of NMOS transistor as it is shown in Figure 1.3 (a). It loses energy due to the interaction with the bounded electrons, causing an ionization of the material it passes through and forming a certain track. Electron-hole pairs are produced along the track of the ion proportional to the ion energy and the length of its track. If there is no voltage applied to the drain the electrons and holes will

recombine. On the other hand, sine the transistor is ON, the voltage applied to the drain separates the electrons and holes. Hence, the electrons are collected by the drain and the holes are collected by the (p+) substrate and a prompt component of current at the drain is observed in the shape of negative pulse as it is shown in Figure 1.3 (b). If the resulting charge of this prompt current is high enough, it will lead to discharging the voltage at the drain for a very short period of time on the order of 100 to 200 picoseconds and will make the transistor OFF at that period of time.

To carry on illustration, suppose that (n2) in the memory cell shown in Figure 1.4 is hit by ion particle. Given that the amount of charge resulting from the induced current by the particle is high enough to switch (n2) from OFF to ON, the memory cell operation will toggle (n1), (p1) and (p2) to OFF, ON and OFF, respectively. Eventually, the bit value stored in the memory cell will change to its opposite value [10, 11].

In the previous generation of CMOS technologies, such a short lived current is not considered in the calculations related to CMOS devices because the sizes of the CMOS transistors is considered respectively large and the resulting prompted current is no more than a small, not apparent, attenuation and usually neglected. With the feature sizes reaching below 0.35 microns, SET and SEU faults are no longer considered a small attenuation. Instead they will be considered normal circuit signals. Moreover, multi ion noise might affect more than one drain of CMOS devices due to nuclear reaction properties of the materials used to manufacture

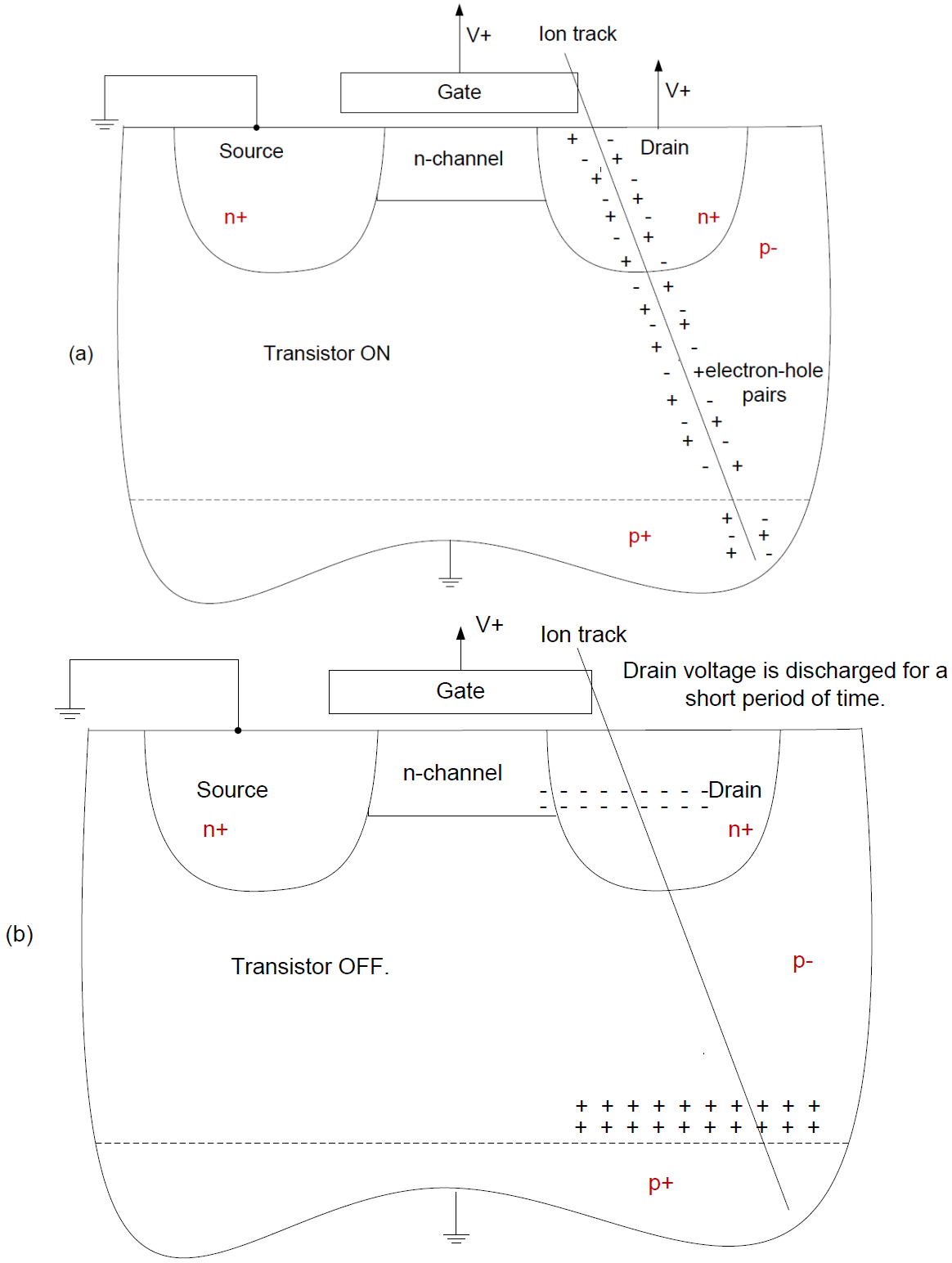


Figure 1.3: NMOS transistor hit by ion particle.

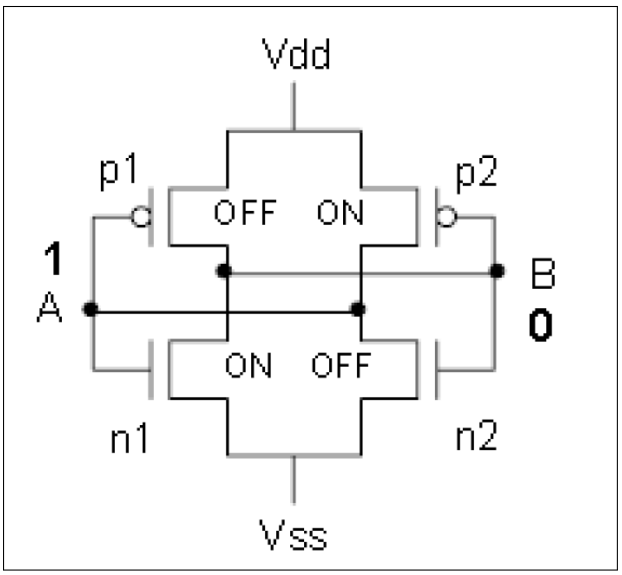


Figure 1.4: Memory cell.

CMOS devices [12]. This will cause serious, if not grave implications for circuits, especially sequential circuits.

In order to give insight about how dangerous this phenomena is, suppose that a sequential circuit is built using nanotechnology using 0.35 micron or even smaller. Assume that the FSM of the circuit require 3 flip flops. Suppose one of the states = (000) gives a read control signal and another state = (001) gives the write signal to the RAM. If a transient fault existed in the circuit specifically in the first flip flop when it was in state = (000), this produces a write command to the RAM instead of a read command. This will cause unwanted overwrite in an inappropriate time which might lead to unexpected behavior of the system.

Fortunately, there are some masking properties that prevent transient faults from affecting sequential circuits, namely: *logical masking*, *electrical masking* and *latching window masking*. Logical masking prevents the SET from propagating

from the fault location to the Flip Flop (FF) by the logic of the circuit. For example, a 2-input AND gate can mask any fault in one input if the other input is grounded. Electrical masking attenuates or completely masks the SET signal due to electrical properties of gates. Latching window masking prevents SET form being captured because it violates the setup and hold conditions of the memory element. Thus the fault won't be latched.

However, the reduction in feature sizes limits the effect of those masking. Also, increasing the clock rates causes the setup and hold times to get shrinked which will limit the latch window masking effect as well.

In order to estimate how effective a circuit can be toward soft errors, soft error rate (SER) estimation researches have been proposed. They fall into three categories: *circuit level*, which computes the probability of a SEU producing an error on the output of a gate hit by ion particle. These techniques use SPICE simulation to obtain these probabilities. *Gate-level* SER estimation techniques try to compute the SER rate of a circuit node by computing the SEU occurrence rate, the error propagation probability (EPP), and the error latching probability. To compute the error susceptibility of a node to SEUs, it is required to compute the probability that the node is functionally sensitized by the input vectors to propagate the erroneous value from the error site to system outputs. *Architectural level* SER estimation is based on electrical and latching-window masking, which provides combinational logic with a form of natural protection against soft errors. Usually Architectural Vulnerability Factor (AVF), which expresses the

probability that a fault in that particular structure will result in an error, is used in architectural level SER estimation.

1.3 Research Motivation

With technology advancement (90 nm, 65 nm, 35 nm and even smaller), systems became more subjected to higher susceptibility to soft errors due to the exponential decrease in device feature size. Therefore, soft errors induced by ion particles are no longer limited to specific field such as aerospace applications and can no longer be neglected. This raises the challenge to come up with techniques to tackle transient or soft faults effects not only in space application but in both combinational and sequential circuits in general. Many researchers have studied the reliability of circuits using some techniques such as circuit level hardening, hardware redundancy and time redundancy. Their main concern is how to avoid the effect of SET and/or SEU. Most of the techniques target the implementation level of a circuit and offer area or time overhead for enhancing reliability. Since steady state probability for several sequential benchmark circuits vary, our method takes advantage of that and chooses some of the states with high probability of occurrence for protection using redundant states. This way the area overhead is kept minimal.

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 sequential circuit, we would like to increase its reliability against transient errors with minimum area overhead and operation cycle time.*

1.4.2 Thesis Objectives

The objective of this work is to investigate the design of soft error tolerant sequential circuits based on adding redundant states at the state diagram level. The objective of adding redundant states is to guarantee tolerance of all single soft errors of states with high probability of occurrence. The minimum number of redundant states will be added to minimize the hardware overhead. State assignment for enhancing soft error reliability will also be investigated. The main project objectives can be summarized as:

- Design of a soft error tolerant technique for sequential circuits at the state diagram level or design level.
- Investigate the impact of state assignment on soft error tolerance and develop and implement a heuristic for enhancing soft error tolerance of sequential circuits.

1.5 Thesis Contributions

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

- Implement a tool for computing state probabilities for a finite state machine.
- Implement a tool for computing soft error reliability for sequential circuits based on Monte Carlo simulation [13]. The objective of this tool is to find the failure rate of a sequential circuit as more faults are observed in the circuit. The monte carlo based simulation tool will be developed using Perl [14–16] in Linux [17–20] system to assess the soft error reliability of the resulting synthesized circuits. (SIS) tool [21] will be used for synthesis and (hope) tool [22] will be used for simulation purposes.
- Develop and implement an algorithm to enhance reliability of sequential circuits based on adding redundant states at the state diagram level to ensure single fault tolerance. The development of the algorithm is done by modifying the finite state machine of a sequential circuit by adding the minimum necessary equivalent redundant states to selected states for which soft error tolerance is desired. Also, the minimum number of bits used in state encoding that ensure fault tolerance of single faults will be computed. It is done using C# [23].
- Evaluate the proposed soft error tolerant design algorithm in terms of circuit reliability and other parameters including area and delay and compare with

existing techniques. A tool will be developed for assigning codes to states according to certain developed heuristics. (SIS) tool will be used in the synthesis process and the developed monte carol based simulation tool will be used to assess soft error tolerance of synthesized circuits.

- Investigate the impact of state assignment on sequential circuits reliability.
- Develop and implement an algorithm for state assignment to enhance soft error tolerance on top of an area optimized design.
- Evaluate the proposed state assignment for soft error tolerance in terms of soft error reliability and other parameters including area.

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 including triple modular redundancy (TMR), hardening techniques, error correction using checkers, selective protection after measuring the soft error effect using probabilistic methods, and some specialized techniques. In addition, a classification of those current techniques will be shown based on the level of abstract that they are targeting.

Chapter 3 proposes a method, namely Algorithm 1, that enhances the fault tolerance of sequential circuit using state redundancy. This chapter starts with

defining state types that are going to be used through out the rest of the thesis. Then, state probability calculation is going to be studied specifically for benchmark circuits to identify the most probable states in each of them. States with the most probability of occurrence are chosen to build the proposed solution. Sequential circuit model in presence of redundant states, the mechanisms of how the solution works, and how it is implemented, are described. Furthermore, we will look into how to calculate the minimum number of bits used in the proposed method, its complexity, and the correlation between state probability and failure rate for a chosen circuit. After that, a comparison between this solution and other well known techniques is introduced in terms of approaching the problem and enhancing fault tolerance. The finishing point of this chapter is reached by giving a conclusion that summarizes the framework of the proposed algorithm.

A novel idea, Algorithm 2, to enhancing reliability to area optimized sequential circuits is discussed in chapter 4. First, the impact of state assignment on sequential circuit reliability is going to be discussed. Second, we will visit how optimizing the area of a sequential circuit is done without taking into account increasing the fault tolerance against soft faults. Third, we will add soft error protection by introducing redundancy to area optimized sequential circuits. The mechanism of the algorithm as well as its complexity are further discussed. Conclusion is given at the end of this chapter to summarize ideas presented in Algorithm 2.

In order to simulate and evaluate the ideas and algorithms which are proposed in Chapter 3 and Chapter 4, Chapter 5 describes the simulation framework and

the method used to evaluate circuit failure probability and reliability as well as the assumptions made in the simulation, tools used to perform it, and fault injection mechanism that is used as a part of the simulation.

Chapter 6 presents a discussion of the proposed algorithms as well as a comparison between them and the current methods regarding fault tolerance. They are done in terms of failure rate, area, and limitations. It also presents many examples of implementing the solution on several sequential circuits benchmarks of ISCAS89 presented as FSM or state table.

The thesis finally concludes in Chapter 7, where the proposed solutions are summarized, with a list of potential improvements and future work.

CHAPTER 2

LITERATURE REVIEW

In this chapter, definitions regarding fault tolerance are reviewed. After that, a survey of the current methods to tolerate SEU/SET in sequential circuits are discussed including triple modular redundancy (TMR), hardening techniques, Error correction using checkers, probabilistic methods to estimate soft errors effect, and some techniques specialized for certain applications. Finally, a classification of the known methods to tackle soft error effect to sequential circuit is explored.

2.1 Definitions

Many terms are used to describe when electronic systems fail. In the next few subsections, the definitions of some terms used through out the thesis are defined.

2.1.1 Defects, Faults and Errors

The definitions of defects, errors and faults terms as defined in [24] 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, elctro-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, fab-

rication errors and physical failures are collectively termed as physical faults [24]. 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 α-particle radiation etc.

2.1.2 Defect (or Fault) Models

While analyzing the defect tolerance of a circuit, the effect of defects in the circuit needs to be simulated. The effect of a production defect can be complex. Accurate defect modeling based on layout and geometrical considerations is normally not an option when the effect of production defects is to be analyzed. For this reason, several defect(or fault) models have been proposed at different levels of abstraction. There are many fault models presented in the literature such as *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 our work, we assume that a single stuck-at fault is used. Therefore, we will explain it in this context. For other defect (or fault) models, the interested reader may refer to [24].

Stuck-at defect (or fault) model is based on assigning a fixed (0 or 1) value to a signal line in the circuit. A signal line is an input or an output of a logic gate or a flip-flop. The most popular forms are the single stuck-at faults which are also known as stuck-at-1 and stuck-at-0 faults.

However, in the simulation part of our work, in order to reflect a temporal fault, an emulation to the stuck-at fault for only 1 cycle time is injected in the faulty circuit.

2.1.3 Failure Rate

Failure rate represents the frequency with which a component of the circuit fails. The failure rate of a circuit usually depends on time, with the rate varying over the life cycle of the circuit. The failure rate 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 measurement interval under stated conditions [25, 26]. In our case the failure rate is redefined as: the total number of failed experiments divided by the total number of performed experiments.

Particularly in semiconductor industry, circuits failure rates can be expressed as failures per billion. The Failures In Time (FIT) rate of a device is the number of failures that can be expected in one billion (10^9) device-hours of operation. That means 1000 devices for 1 million hours, or 1 million devices for 1000 hours each, or some other combination [27–29]. One calculation made by a nice white paper with many references concludes that 1000 to 5000 FIT per Mbit (0.21 error

per day per Gbyte) is a typical DRAM soft error failure rate [30].

2.1.4 Reliability

The reliability of a system can be defined as the ability to perform the specified function under stated conditions [31]. 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. If the failure rate of the system is constant over time, the reliability function is $R(t) = e^{\lambda\ t}$ where λ is the constant failure rate for one unit of time. When $\lambda\ t$ is small, $R(t) \approx\ 1 - \lambda\ t$.

A defect-tolerant system can continue to operate despite a certain number of defects. For such systems, where not all subcomponents need to be working, more elaborate reliability calculations need to be performed. In order to achieve that, Monte Carlo simulations need to be employed [13].

An alternative evaluation criterion is Mean Time To Failure (MTTF) which is the average time a system will run before failing. MTTF is linked to the failure rate in the following way: $MTTF = (1/\lambda)$. If λ is the failure rate per hour, MTTF is the average number of hours before failing. When considering how reliable a system is in the presence of production defects, time is not relevant. MTTF is therefore not applicable and reliability is simply $R = (1 - \lambda)$ where λ is the probability of failing under stated conditions [26].

2.1.5 Fault Tolerance

The term fault-tolerance first appeared in technical literature in 1967, defined as: A system is fault-tolerant if its programs can be properly executed despite the occurrence of logic faults [32]. The prime motivation at that time was challenges in interplanetary exploration. The objective of fault-tolerance is either to mask, or to recover from, faults once they have been detected [33]. It is in contrast to the second method of achieving system reliability, fault prevention, which seeks to eliminate all faults before the system is put to use. Complete fault prevention is impossible to achieve in practice and high degrees of prevention is costly [34]. Fault-tolerance is therefore commonly used in increasing system reliability, often in combination with partial fault prevention. Much research on reliable systems is concerned with the detection of faults using error detecting and correcting codes or fault-detecting and self-repairing circuits. The tolerance itself is achieved using redundancy techniques. Those techniques are going to be discussed in the next section.

2.2 Current Fault Tolerance Methods

In order to overcome the soft errors effect problem in sequential circuits, several techniques have been introduced. Some of these techniques are based on tolerating errors in combinational circuits while other techniques are focused on tolerating errors occurring at memory elements. Most of these techniques are applied at the gate/ implementation level of a circuit. Also there are several error correcting

mechanisms using checkers presented in this field. Moreover, temporal faults are estimated and modeled using probabilistic methods.

2.2.1 Triple Modular Redundancy (TMR)

Triple Modular Redundancy (TMR) is one of the famous techniques to reduce the impact of hard/soft errors in combinational logic. It is one of the generalized 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 identical modules whose outputs are voted on. If 2 modules out of 3 produce expected correct results, this implies that the majority of the modules produce correct results, and hence the system will be functional. In TMR, all the three identical modules perform the same operation, and a voter accepts outputs from all three modules, producing a majority vote at its output as shown in Figure 2.1 [35].

In such a structure, M = 2 and N = 3 and a 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), making the approach difficult in the context of highly integrated nano-systems [36]. Despite this limitation, TMR is heavily used in practice whenever the reliability of the circuit is a crucial demand especially when single faults are needed to be protected.

In sequential circuits, TMR voting idea is implemented in similar way. In [10,

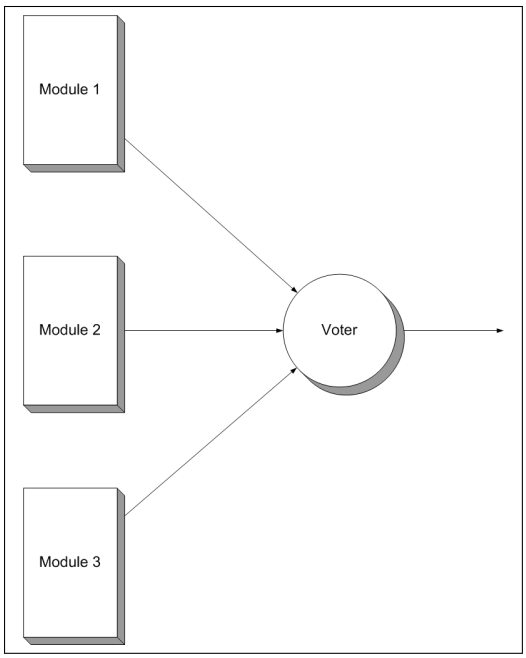


Figure 2.1: A Triple Modular Redundant (TMR) structure.

11] a TMR hardening technique addressing SEU faults is proposed based on a temporally redundant sampling latch shown in Figure 2.2. The temporal latch approach provides immunity to soft errors. The latch is replicated three times to generate samples of output in different clock cycles and then a majority voting mechanism is used to prevent soft errors from propagating. It is suggested that every flip flop be replaced by this temporal latch.

The proposed temporal sampling latch is shown in Figure 2.2. The circuit contains nine level sensitive latches (U1 through U9), one majority gate (U10), and three inverters. Two level sensitive latches in tandem and clocked by complementary clock signals (such as U1 followed by U2) form an edge triggered D-Flip-Flop. With the clock inversions used in Figure 2.2, the D-Flip-Flops formed by (U1, U2), (U3, U4), and (U5, U6) are triggered on the falling edges of the clocks CLKA,

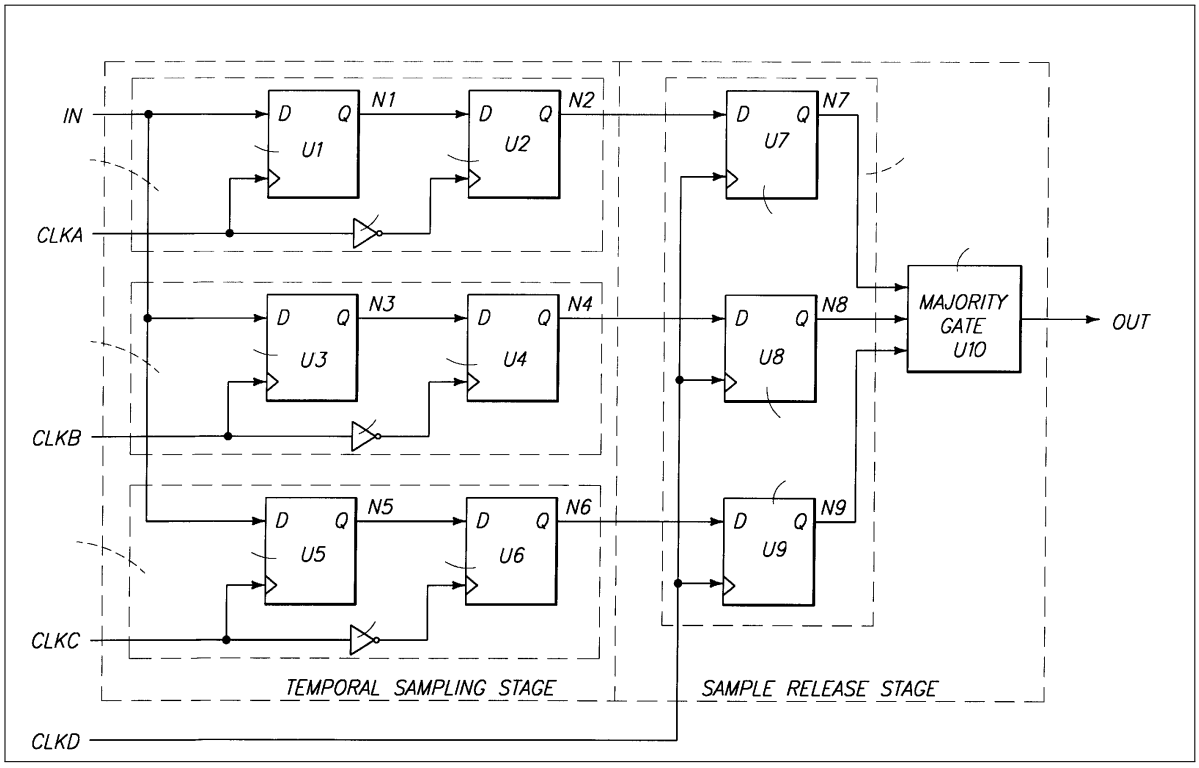


Figure 2.2: Temporal sampling latch with sample and release stages.

CLKB, and CLKC, respectively. The three D-Flip-Flops (U1, U2), (U3, U4), and (U5, U6) operate in parallel and form the temporal sampling stage of the circuit. Each of these three D-Flip-Flops drives another level sensitive latch. These latches (U7, U8, and U9) together with a simple majority gate (U10) form the sample release stage of the circuit.

The combinatorial logic is effectively replicated, not in space but in time. The same logic is really just used at three different times. The result of this is that errors are flushed on each clock cycle and the maximum error latency never exceeds a clock period. Temporal redundancy of the circuit is achieved by combining a temporal sampling stage with a sample release stage where the sampling is controlled by three clocks (CLKA, CLKB, and CLKC) and the release

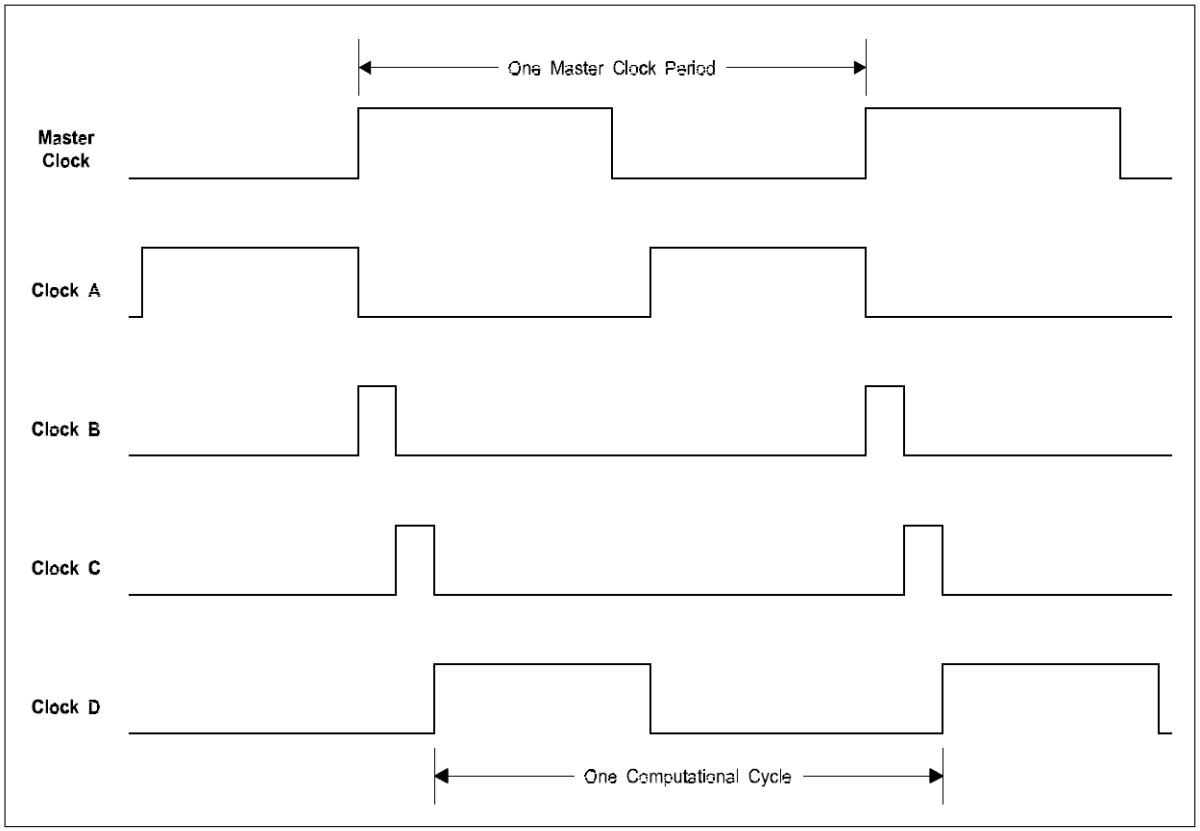


Figure 2.3: Temporal latch control clocks derived from master clock.

is invoked by CLKD. The clocking scheme is therefore central to the operation of circuit.

The clocking scheme is shown in Figure 2.3. The figure shows two cycles of the master clock and two cycles of the temporal sampling latch control clocks. The master clock (top curve) would generally be the clock signal brought onto the chip through an input pad. The bottom four curves show the four clock signals (CLKA, CLKB, CLKC, and CLKD) used above in Figure 2.2. The CLKB and CLKC widths need only be greater than the maximum width of any SET induced in the combinatorial logic.

The operation of the circuit of Figure 2.2 with the clocking sequence of Fig-

ure 2.3 is most easily explained if we start at the beginning of a computational cycle which begins at the rising edge of CLKD. At this time the sample release latches (U7, U8, U9) pass their input data to the majority gate (U10) where it subsequently appears at the output node (OUT). CLKD subsequently goes low, the release latches (U7, U8, U9) enter a hold state, and this original data remains asserted on the output for the remainder of the computational cycle.

This output data is then processed by intervening combinatorial logic before it appears at the input to the next temporal sampling latch. The data must arrive at the input to the next stage before the falling edge of CLKA at which time the data is stored in the a D-Flip-Flop formed by latches U1 and U2. CLKB then goes high to sample the input. Whatever data is at the input when CLKB goes back low is then stored in the D-Flip-Flop formed by latches U3 and U4. Finally, CLKC toggles high and low to sample and hold the input data in the final D-Flip-Flop formed by latches U5 and U6. At this time another computational cycle begins. The input data to each temporal sampling latch has been asserted on the three inputs to the sample release stage. When this next computational cycle begins, CLKD again goes high and the data appears at the output of the majority gate. Data is released to the combinatorial logic on each rising edge of CLKD and must reach the next sampling latch before the falling edge of CLKA (minus the setup time). From Figure 2.3, this is thus the period of the master clock minus the D-Flip-Flop setup time and minus the sum of the CLKB and CLKC durations. An equivalent way of viewing the incurred speed penalty is that the temporal

latch setup time is effectively increased by the sum of the widths of the CLKB and CLKC phases.

As it is shown in Figure 2.2, the Majority gate again (i.e. voter) causes the same limitation which is that it can not protect against soft errors happening in it.

In [37], the same idea is presented. A systematic error masking technique based on using latch majority voting to tackle soft errors in combinational and sequential circuits is discussed. It is based on studying the latching windows and introducing a method that uses temporal redundancy. It prevents a SET pulse of width less than approximately half of the slack available in the propagation path from latching and turning into a soft error.

2.2.2 Latch Hardening Techniques

Although TMR for sequential circuits described in the previous section is basically one type of latch hardening, this section is exploring different methods that involve a change in the latch design to achieve soft error tolerance.

In [38], the authors review several designs and compare them based on their robustness and their power and performance overheads. They also propose new latch designs, the best of which is vulnerable only to a single-event, multiple-upset without any delay overhead and consumes only 40% power of a standard latch. This new latch can be applied as input to a sampling circuit which drives a voting circuit to get a tolerated output.

The reviewed latch designs are: *single event upset (SEU) tolerant latch* depicted in Figure 2.4, *soft-error hardened latch* depicted in Figure 2.5 and *dual interlocked storage cell(DICE)* depicted in Figure 2.6. In addition, the authors in [38] *proposed latch* is depicted in Figure 2.7.

The schematic of the latch proposed in [39], is shown in Figure 2.4. The latch stores data D at PP and NN while complement of data (D) is stored at QP and QN. PP and QP are driven only by PMOS transistors while NN and QN are driven only by NMOS transistors. This latch utilizes the fact that only a 0 to 1 flip can occur in a PMOS and only a 1 to 0 flip can occur in an NMOS, due to which nodes QP and QN are soft-error hardened. Particle strikes at any of the two nodes PP or NN do not change the output Q, however they cause node Q to become dynamic for a short period of time. To see more details for this latch and the operation of other latches, the reader can find more information in [39], [40], [41, 42] and [38], respectively.

The latch shown in Figure 2.4 can be used to prevent soft errors due to particle strikes in combinational logic. This requires inputs DP and DN to be fed from two different *Combinational Logic Block* (CLB)s or delayed by certain time interval. The delay of this latch was found to be 5 times of the original latch due to the high CK-Q delay. The latch can be assumed to be *Single Event Multiple Upset* (SEMU) hardened for most commodity applications. This latch consumes static power due to some of the PMOS and NMOS transistors not completely turning off. The power consumed by the latch is found to be 1.52 times of the original

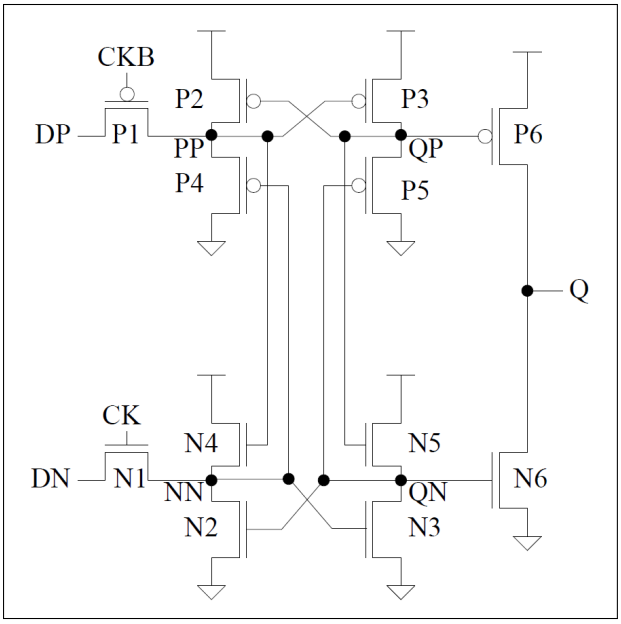


Figure 2.4: Single-event-upset-tolerant latch.

latch. Clearly the number of transistors used in this design is 12.

The latch shown in Figure 2.5 can only handle SEU on the latch itself. Any transient pulse with width equal to the latching window of the modified latch can cause soft error if it is not logically or latching window masked. However, soft errors can be caused due to SEMUs on the latch. The delay overhead of this latch was reported to be 5-20% of the original latch. The power consumed by the latch was reported to be just 2-3% of the original latch. The number of transistors used in this design is 18.

The latch shown in Figure 2.6 does not mask transient pulses originating in CLB. This latch is also highly vulnerable to SEMUs as compared to other latches analyzed before. The delay of this latch was reported to be similar or just 2-3% higher than the original latch. The power consumed by the latch was reported to be just 34% of the original latch. The number of transistors used in this design is

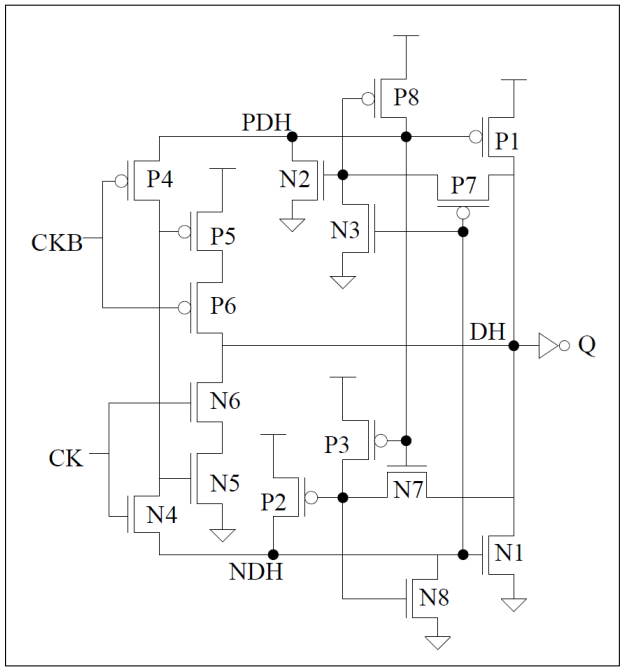


Figure 2.5: Soft-error hardened latch.

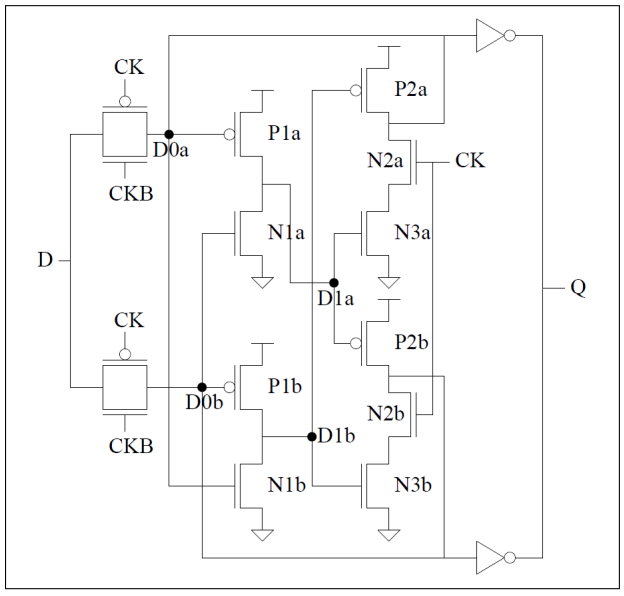


Figure 2.6: Dual interlocked storage cell(DICE).

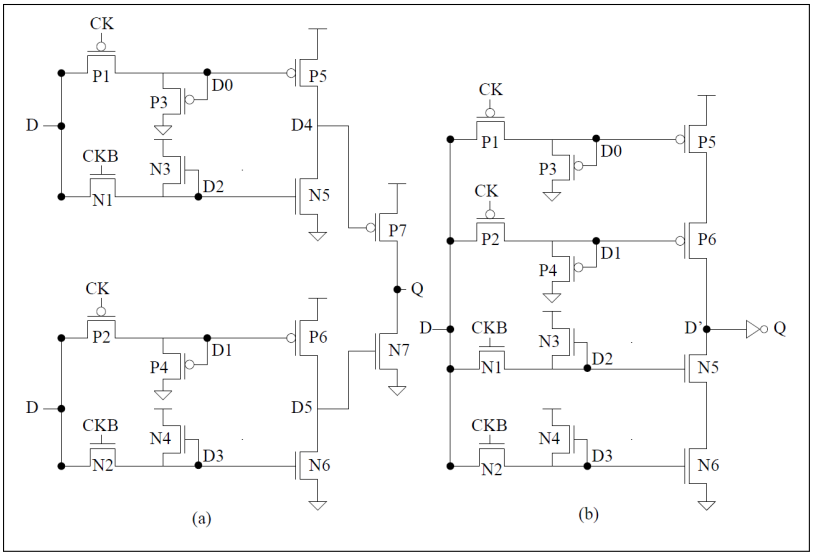


Figure 2.7: (a) Latch A having best power, performance and soft-error immunity among the proposed latches. (b) Latch B can be used to provide soft-error protection for CLBs also.

18.

Advanced and robust designs latches are depicted in Figure 2.7. The latch in Figure 2.7(a) doesn't have SEMU immunity in most of the cases. However, the delay is almost the same as the original latch and the power consumption is only 40% compared to original latch. An alternative design shown in Figure 2.7(b) gives better immunity to SEMU and provides soft-error protection for transient faults in CLB. Moreover, the delay is almost twice as the original latch and the power consumption is only 30% compared to original latch.

In [28], it is concluded that 49% of the overall SER for a design manufactured using state of the art technologies results from sequential elements. They also propose scan reuse method in which they reuse the scan portion in scan cell design to reduce the impact of soft errors by trapping the error inside the scan

path and then shifting out within a specific number of cycles. This technique involves adding a small amount of gates to the scan cell design . However, it uses operation cycles to recover the circuit operation.

To look at it in details, Figure 2.8 shows a microprocessor scan flip-flop design that comprises two distinct circuits: a system flip-flop and a scan portion. All scan flip-flops in a design are connected together as one or more shift registers. The SI input of a scan flip-flop is connected to the SO output of the preceding scan flipflop in the shift register. The structure of the scan portion of Figure 2.8 is similar to the system flip-flop, with the addition of interface circuits to move data between the system flip-flop and the scan portion, as well as shifting the test pattern and test response, as required by the specific scan architecture. This design has two operation modes: normal-system operation and test. In the test mode, clocks SCA and SCB are applied alternately to shift a test pattern into latches LA and LB. Next, the UPDATE clock is applied to move the contents of LB to PH1. Thus, a test pattern is written into the system flip-flop. Next, functional clock CLK is applied, which captures the system response to the test pattern. Finally, the CAPTURE signal is applied to move the contents of PH1 to LA. The system response is then shifted out by alternately applying clocks SCA and SCB. During normal system operation, the scan portion is shut off by asserting logic-0 values to the scan signals (SCA, SCB, UPDATE, and CAPTURE). There are three basic reasons for using the scan style of Figure 2.8: structural testing using automated test pattern generation tools, functional testing using signature

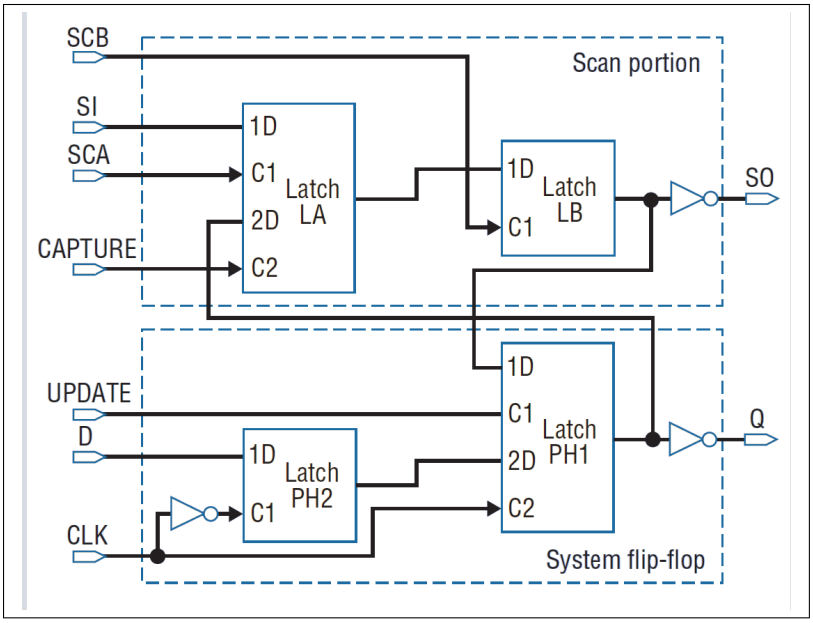


Figure 2.8: Microprocessor scan cell design.

analysis, and efficient post silicon debug. The opportunity for scan reuse for soft-error protection arises from the redundant scan resources, latches LA and LB in Figure 2.8, that are unused during normal operation, but add to the occupied area of the chip and the leakage power during normal operation.

Figure 2.9 shows how reusing the scan flip-flop design can reduce the impact of soft errors that affect latches. The flip-flop designs test mode operation is identical to the design in Figure 2.8. In normal system operation mode, the scan clocks SCA, SCB, UPDATE, and TEST are forced low, while the CAPTURE signal is forced high. This converts the scan portion into a master-slave flip-flop that operates as a shadow of the system flip-flop. During normal operation, when the clock signal CLK is 0, the C-element output drives flip-flop output Q, and the chip transfers the logic value at input D into latches LA and PH2. During this time, latches PH1 and LB are susceptible to soft errors because their clock inputs

are 0 and they are holding logic values. If a soft error occurs in PH1 or LB, the logic value on O1 will not agree with O2. As a result, the error will not propagate to output Q, and the keeper will hold the correct logic value at Q. A soft error in PH2 or LA when CLK = 1 produces similar results. Depending on the systems speed and the leakage current, the keeper in Figure 2.9 might not be necessary. Extensive SER simulations on an advanced process technology using an internal tool show that this design can reduce the SER by more than 20 times compared to the error rate for an unprotected flip-flop. Any soft error affecting a single latch inside a flip-flop is guaranteed to be detected by a self checking scan flip-flop that is obtained by removing the C-element and the associated keeper structure from the design in Figure 2.9. Various self checking scan cells choices are possible. During normal operation, at least one copy of correct data exists, under the assumption of a single error in a latch. To perform self-checking, the approach implements error-detection circuits such as equality checkers that compare the Q and Q2 outputs of all such flip-flops in a design and indicate an error each time a mismatch occurs. A major drawback of such self-checking approach is the significant amount of area occupied by the logic network that accumulates the error signals generated by individual flip-flops and produces one or more global error signals.

The error-trapping scan cell shown in Figure 2.10 eliminates this problem. Latches LA and LB store redundant copies of the PH2 and PH1 content, respectively, during normal operation. A soft error in any latch causes the error signal (E) to be 1. This signal drives the top input of the exclusive-or gate XOR2 so that

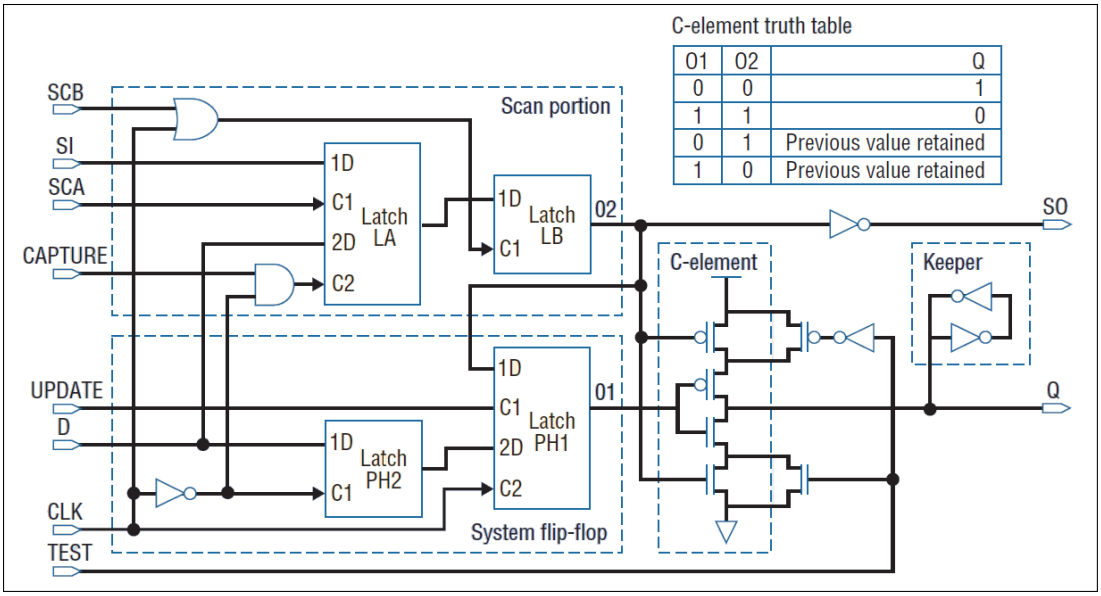


Figure 2.9: Scan reuse.

when E equals 1, the output of XOR2 (D1) becomes the complement of D. Once the error signal E is 1, the logic values stored in LA and LB become complements of the contents of PH2 and PH1, respectively, and E continues to be 1. Thus, the error is trapped until another soft error affects one of the latches of this flip-flop, which is a rare event. After a pre-specified number of clock cycles, at a recovery point the system shifts out this trapped error signal using the existing scan path, which eliminates the need for global routing of error signals at the cost of error-detection latency. Re-execution then achieves error correction. It is shown by simulation that there is high gain in soft-error resilience with relatively low overheads, which justify the use of proposed designs in various applications. Several optimizations are possible to further reduce the system-level power overhead to 3 percent or less. The reuse paradigm for built-in soft-error resilience offers the following unique advantages over existing soft-error protection techniques: mini-

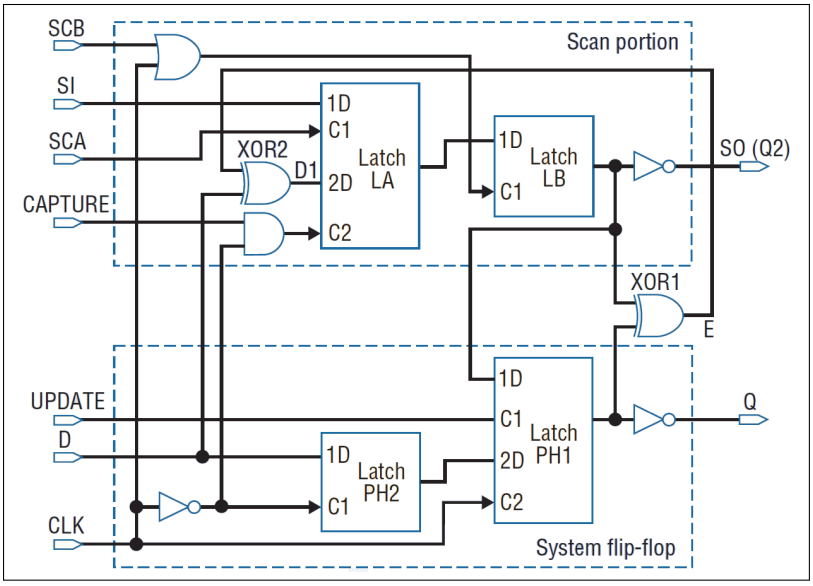


Figure 2.10: Error trapping scan cell design.

mal area overhead because resources already available for test and debug can be reused for soft-error resilience, minimal routing overhead, no major architectural changes required, applicability to any design including microprocessors, network processors, ASICs, and a broad spectrum of design choices with several area, power, performance, and soft-error rate trade offs. For example, the design shown in Figure 2.9 can be redesigned to achieve a 50 percent rather than a 20 times reduction in the SER, with a 30 percent reduction in the cell level power overhead.

2.2.3 Error Detection using Checkers

The general idea for achieving error detection and correction is to add some redundancy (i.e., some extra bits) to a message, which afterward can be used to check consistency of the output bits, and to recover bits determined erroneous. Checkers used this concept to detect if an output codeword is correct and correct

it if possible [43].

In [44], a novel reduced m out of n coding method is proposed that can be used for the synthesis of a totally self checker for tolerating soft errors. The authors propose a totally self-checking synchronous sequential circuits that are able to recover after an occurrence of a fault, they call the ability of such circuits *self healing ability*. The novel reduced m-out-of-n code is based on a Look Up Table implementation of monotonic functions. For most circuits in a standard benchmark set, the proposed approach leads to a reduction of about 10-20% of the overhead as compared with traditional methods.

Three operation modes of the of sequential circuit taking into account that transient fault might occur are presented as shown in Figure 2.11. They are: *free mode* (F), when no faults are observed in the circuit, *Erroneous mode* (E), when there is a fault that produces non-code output vector (i.e. wrong output), *silent mode* (S), if a non-code next state vector is produced while the output vector is a codeword.

To illustrate their idea, we consider that two code variables Z_i and Z_j are compatible if they are both equal to 1 in at least one codeword, otherwise they are incompatible. Their idea is based on constructing the reduced m-out-of-n code by grouping incompatible output code variables in the same subset. The resulting subsets form a one-to-one correspondence to a specific check bit c_i. Check bit c_i is assigned to 1 only if all of their corresponding output code variables equal to 1.

For instance, Table 2.1 shows a reduced codes example. The partitions $\{\{Z_1,$

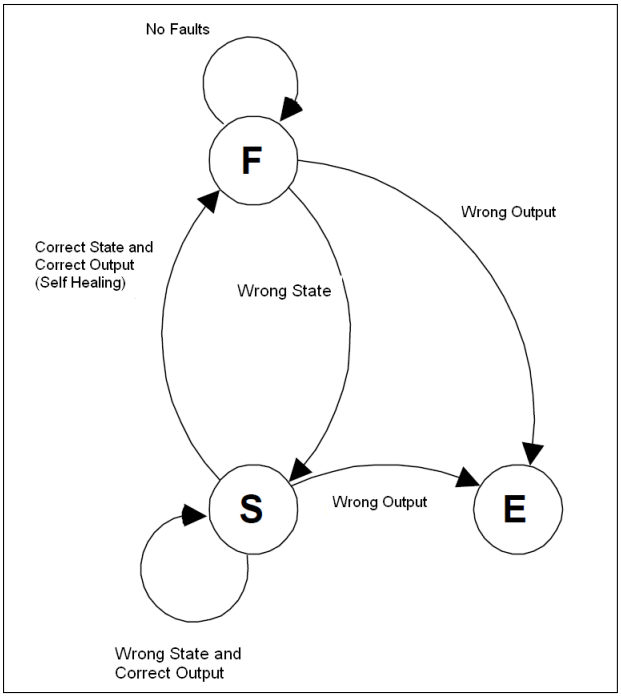


Figure 2.11: Mode transition graph for a transient fault.

Z_3, Z_4}, {Z_2} , {Z_5} } have been constructed based on compatibility condition. For example, Z_1 and Z_3 are not compatible because both of them are never equal to 1 in all of the codewords (i.e. rows). Check bits are represented by three right hand columns in Table 2.1. They can be obtained by $c_1 = \text{NAND}(Z_1, Z_3, Z_4)$, $c_2 = \text{NOT}(Z_2)$ and $c_3 = \text{NOT}(Z_5)$. The coding partition {{b_1, b_3, b_4, b_6}, {b_2, b_7} , {b_5, b_8} } corresponds to the reduced 3-out-of-8 code. Using such partitioning, a Look-Up-Table-based (LUT-based) FPGA is designed so that it detects errors on the set of acceptable codewords listed in Table 2.1. Figure 2.12 shows 8-input LUT-based FBGA block diagram that is used in the scheme where the corresponding functions are indicated. The checker detects errors according to two signals, R_1 and R_2. If $R_1 \neq R_2$, the scheme is fault free, otherwise a fault has occurred.

For example, given the codeword 11001000, the checker's intermediate vari-

Table 2.1: Reduced codes example.

Output code words	Information bits					Check bits		
	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8
	Z_1	Z_2	Z_3	Z_4	Z_5	c_1	c_2	c_3
o_1	0	0	0	0	0	1	1	1
o_2	1	1	0	0	1	0	0	0
o_3	0	1	0	1	0	0	0	1
o_4	0	0	1	0	1	0	1	0
o_5	0	0	1	0	0	0	1	1
o_6	1	1	0	0	0	0	0	1
o_7	0	1	0	0	0	1	0	1

ables are $\phi_1 = 1$, $\phi_2 = 0$, $\phi_3 = 1$ and $\phi_4 = 0$. Hence, $R_1 = 1$ and $R_2 = 0$. Therefore, $R_1 \neq R_2$ and the codeword is fault free and is similar to o_2. If the 4^{th} bit is in error, the codeword will be 11011000. The checker's intermediate variables are $\phi_1 = 1$, $\phi_2 = 1$, $\phi_3 = 1$ and $\phi_4 = 1$. Hence, $R_1 = 1$ and $R_2 = 1$. Therefore, $R_1 = R_2$ and the codeword is faulty.

Experimental results show that such an encoding gives better results than Berger output encoding [45] and the Smith output encoding [46, 47] in terms of area overhead.

2.2.4 Soft Error Effect Measurement Using Probabilistic Models

Probabilistic models are often used to measure the effect of soft errors in systems. The purpose of soft error estimation models is to measure the importance of combinational and sequential parts of a given circuit. Hence, techniques that

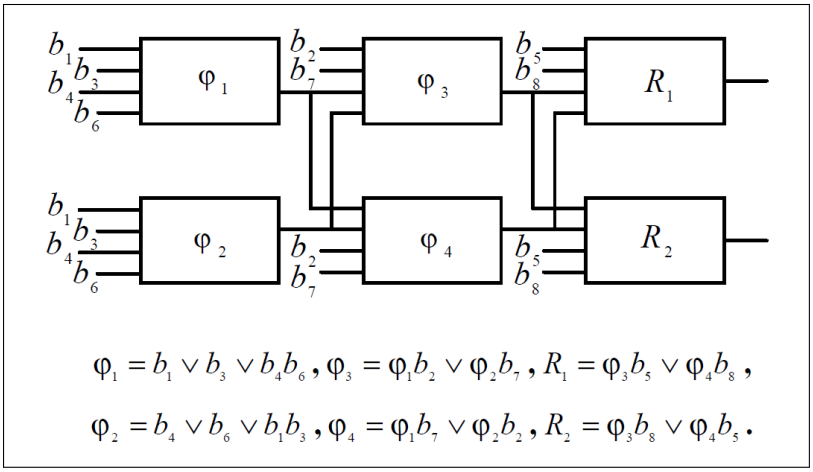


Figure 2.12: Block diagram of the 3-out-8 checker example.

tackle soft errors are used to selectively harden or protect them.

In [48], an analytical framework to analyze multi-cycle error propagation behavior is used and then system flip flops are ranked based on their effects on system level soft error rate. Using this ranking, flip flops are protected selectively against soft errors. Soft errors due to single event upsets are the main reliability threat for digital systems. They have developed a mathematical framework for the estimation of system failure probability in multiple cycles after the SEU event. By computing combinational error propagation probabilities between bistables, they have devised a matrix formulation to represent the probability of error in each bistable or primary output at any given clock cycle. Based on this formulation, they have computed mean time to manifest an error (MTTM) for each flip-flop.

In [49], a soft error estimation method is proposed that is faster than the traditional random fault injection methods. A systematic approach for soft error rate estimation is developed. The proposed approach leverages the signal probability calculation, which is already used in other steps of the design flow, and com-

putes the error propagation probability. Some efficient graph-based algorithms have been used for this computation. To improve the accuracy of their approach, they have considered the output dependencies. Experiments on benchmark circuits and comparison of the results with the random simulation technique show the effectiveness and the accuracy of their approach. It is about 95% accurate compared to the random-simulation method while 4-5 orders of magnitude faster. Their analysis helps the designers to evaluate the vulnerability of gate-level designs to soft errors and the contribution of each gate to the overall SER. Gates which are contributing most to the overall system vulnerability can be protected by circuit-level radiation hardening techniques.

In [50], the probability of latching an incorrect value in the system is estimated based on static timing analysis and signal probabilities. They present a very fast and accurate approach based on enhanced static timing analysis and signal probabilities to estimate the probability of latching an incorrect value in the system bistables (timing derating). They have proposed a combined logic and timing derating estimation method in sequential circuits. The proposed technique uses an enhanced static timing analysis to derive all possible erroneous waveforms propagated from a struck gate to reachable flip-flops and calculates the probability of latching an incorrect value in flip flops. They also exploit a technique based on signal probability to estimate propagation probabilities. Experimental results and a comparison with timing accurate Monte-Carlo simulations show that their proposed technique is 4-5 orders of magnitude faster while the difference in

accuracy is almost 1% on average.

In [51], a metric is introduced called the critical soft-error rate (CSER) which covers soft errors that are not masked by the three masking effects, as an alternative to conventional SER. The single stuck at fault model is used to estimate the transient error probability of circuit nodes. Selective hardening is used as a systematic hardening strategy based on transient error probability as a guide to improve soft error tolerance.

In [52], an approach for evaluating soft errors effect using symbolic modeling based on BDDs/ADDs together with probabilistic model is proposed. It is concluded that SER decreases below 10^{-7} FIT (failure in time - one failure in 10^9 hours of operations) within 10 clock cycles after the hit. Hardware unrolling techniques are used to tackle soft errors which introduce more hardware to the system.

Estimation techniques are proposed in [53]. By coupling probability theory with concepts from testing and logic synthesis, they present accurate and scalable algorithms for reliability analysis of logic circuits. They described two accurate, scalable, and highly efficient techniques for reliability analysis of logic circuits. These techniques have several potential applications, including redundancy-free reliability optimization, asymmetric and fine-grained redundancy insertion, and reliability-driven design optimization.

In [54], the authors discussed faults, error bounds and reliability modeling of nanotechnology-based circuits. They briefly review failure mechanisms and

fault models in nanoelectronics. Reliability functions based on probabilistic models are developed for unreliable logic gates. Then, they show that fundamental gate error bounds for general probabilistic computation can be derived using the nonlinear mapping functions constructed from the gate models. An analytical approach based on the probabilistic gate models (PGMs) is proposed for estimating reliabilities of nanoelectronic circuits. This approach is based on the probabilistic modeling of unreliable logic gates and interconnects which is based on the iterative execution of PGMs according to a circuits structure. In spite of the approximations used in probabilistic modeling, their study suggests that the proposed approach provides a simple and efficient way to model the reliability of nanoelectronic circuits.

2.2.5 Special Techniques

Using verification schemes at the algorithmic level, a method is proposed in [55] using Ferivalds technique to minimize soft errors effect in combinational matrix operation. The technique is extended to sequential matrix multiplication. However, this technique is limited to some applications that use matrix operations such as image processing.

2.3 Classification of Current Techniques

The current techniques used to tolerate transient errors fall into 3 major classes: *circuit-level hardening*, *classical hardware redundancy*, and *time redun-*

dancy techniques.

Circuit level hardening methods are based on special circuit level design for combinational or sequential elements. Those techniques can be simply injected in your design by simple replacement and have minimal operation performance overhead. However, it is expected that they introduce area overhead.

Hardware redundancy methods use classical methods such as triple modular redundancy (TMR) and concurrent error detection such as duplication, parity prediction and low cost techniques for matrix operation. Usually it is hard to integrate such methods in old designs. Moreover they introduce hardware redundancy.

Time redundancy approaches use more operation cycles to recover from faults. They can be divided into 3 types: software implemented hardware fault tolerance approaches, Multithreading techniques and multi-strobe methods.

Software implemented hardware fault tolerance approaches take advantage of software redundancy. In those approaches, program instructions are executed twice and the results compared to detect errors. Since the instructions run two times the performance overhead can be more than 100% if we include the time to compare the results. Moreover, they use special control flow checking techniques. On the other hand, no hardware redundancy is presented. Multithreading techniques take advantage of software threads by executing the same instruction sequence using two threads and then compare the results to detect errors. The performance overhead in this type is reduced compared to Software implemented

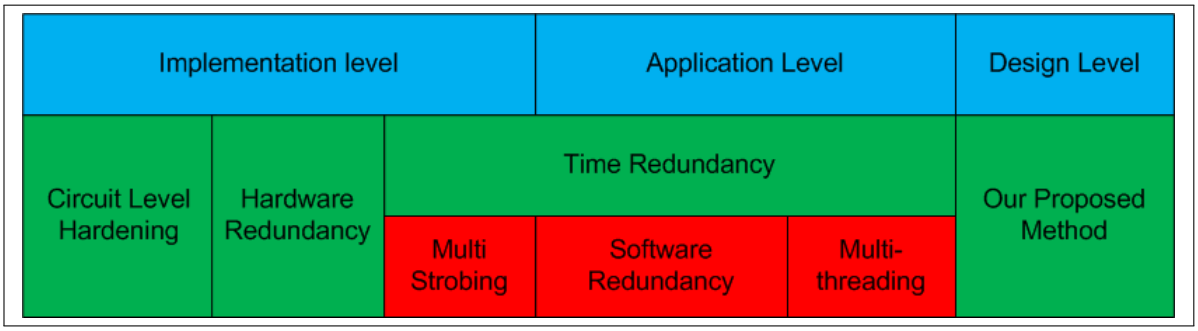


Figure 2.13: Current techniques for soft error tolerance.

hardware fault tolerance approaches. However, it needs high level threading management protocol to organize the operation. Both of software implemented hardware fault tolerance methods as well as multithreading techniques are usually applicable for microprocessors operations only. The last time redundancy technique is multi-strobe methods. The faults are detected and corrected by strobing outputs of the same combinational logic block multiple times separated by delayed clocks. It can be selectively injected in some portion of the design unlike the other 2 methods. In addition, they can be applied to unlimited logic blocks.

Figure 2.13 classifies those different techniques and fit our proposed method in the hierarchy.

CHAPTER 3

FAULT TOLERANCE USING STATE REDUNDANCY: DESIGN LEVEL APPROACH

In this chapter, we will introduce a novel idea to increase sequential circuit reliability and hence fault tolerance by adding redundant equivalent states to the states with high probability of occurrence in sequential circuit behavioral machine. As mentioned in Section 1.1, sequential circuits can be represented by FSM which contains group of states. If a certain number of states is known to be more frequent using probability calculations [56–58], it is wise to enforce more protection upon them by adding redundant states to them. This will increase the reliability of sequential circuit as it it very likely that the error occurs when the circuit is either in one of these states or going to one of them. To maintain the same operation of the unprotected FSM, the newly added redundant states have

the same input, next state, and output as the original protected states.

Therefore, to reduce the soft temporal errors effect, we replicate the designated, to be protected, state with redundant states. The states which have highest probability of occurrence are the ones that we are going to replicate. Other states with low probability are kept without adding redundancy. This way the area overhead is reduced.

According to our knowledge, none of the previous techniques has presented the introduction of redundant states for states with high probability of occurrence at the FSM level. This idea introduces a new class of methods that tackle the soft error effect in early stages, namely the design stage. We can say that, all the techniques mentioned in Sections 2.2 and 2.3 are under implementation level or application level methods. Our idea can be considered a design level method as shown in Figure 2.13.

In the following sections, the type of states that are going to be used through the thesis is described as an introductory. Then, the proposed design level approach is discussed in details. After that, the proposed idea is compared with previous popular methods in terms of how they approach the problem.

3.1 Type of States

The original states of FSM are generated in a way to ensure that the minimum number of states are used to represent the sequential circuits. Our proposed method introduces redundant states to some of the original states that happen

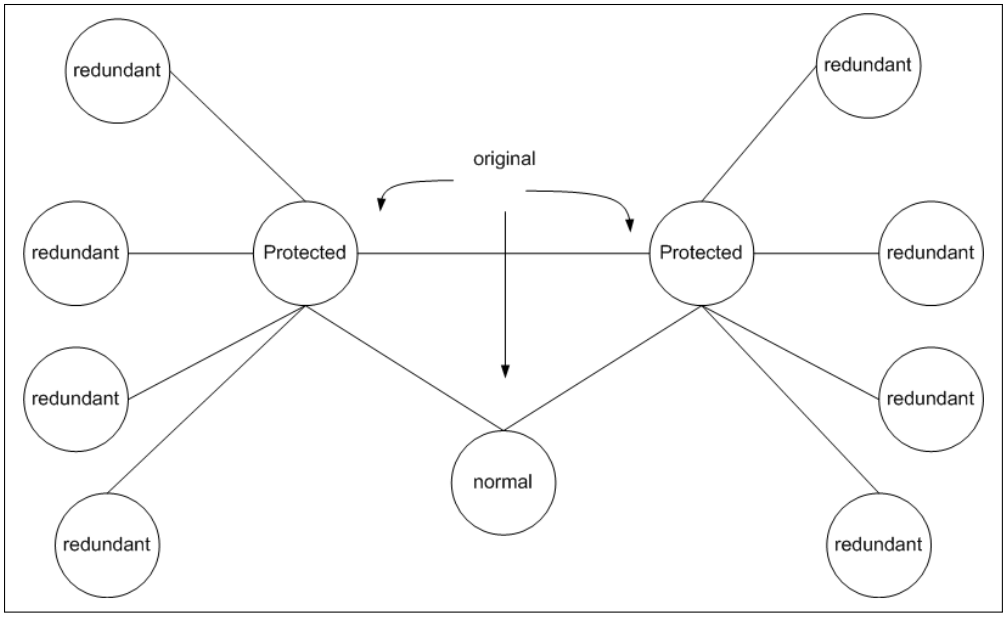


Figure 3.1: State types.

more frequently, so that temporal faults are tolerated. Therefore, we can say that states in a fault tolerant sequential circuit can be classified into 3 types: normal states, protected states, and redundant states. Normal states are original states that are not protected for fault tolerance. Protected states are states that will be considered for reliability enhancement due to their high probability of occurrence. For each protected state, equivalent redundant states will be added to guarantee single soft fault tolerance. Figure 3.1 illustrates an example that shows different types of states. There are 3 original states in Figure 3.1, 2 of them are protected and 1 is normal. Each of the 2 protected states has 4 redundant states. The number of redundant states added will be the minimum number required to satisfy fault tolerance as it will be discussed in Section 3.3.

3.2 State Probability Calculation

In order to decide which states of the original states of a finite state machine need to be protected, states must be sorted based on their probability of occurrence. Probabilistic approaches try to correlate the various probabilities in order to calculate steady state probabilities if the FSM is simulated for infinite amount of time.

Actually, logic synthesis [59], verification [60], testing [61,62] and more recently, low-power design [58, 63–67] have benefited from using probabilistic techniques. In particular, the behavior of FSMs has been investigated using concepts from the Markov chain (MC) theory. Studying the behavior of the MC provides us with different variables of interest for the original FSM. In this direction, [63,66,67] are excellent references where steady-state and transition probabilities (as variables of interest) are estimated for large FSMs. It is also achieved by repeated application of the Chapman-Kolmogorov equations [58, 68].

In our approach, statistical techniques can be fast and accurate since a short representative sequence for an FSM can be determined. Therefore, we perform statistical approaches by simulating the state machine using random provided input vectors and determining the state probabilities based on it. Tables 3.1, 3.2 and 3.3 show several ISCAS89 FSM benchmarks with their states sorted in descending order based on the most frequent state. It is done by simulating the FSMs for about 250000 iteration and recording the counters for each state. The resulting steady state probability is calculated by dividing those counters by 250000 for

each state.

3.3 The Proposed Design Level Approach

In order to reduce the soft temporal errors effect, our design level approach protects the highly probable states by introducing redundant states to them. By keeping other states with low probability without adding redundancy, the area overhead is kept minimal.

To illustrate our idea, sequential circuit model in the presence of redundant states is explored. Then, state encoding using the minimum number of bits is discussed. After that, the proposed design level approach that increases fault tolerance in sequential circuit based on state redundancy is discussed in details.

3.3.1 Sequential Circuit Model in the Presence of Redundant States

Sequential circuit reliability can be increased by adding redundant equivalent states to the FSM. The redundant states have the same input, next state, and output as the protected states. In order to illustrate this let us consider the simple bit flipper example mentioned in Figure 1.2. Given a FSM with two states A and B, let us assume that both states need to be protected. Originally the FSM needs only one D-FF to be implemented. To tackle single errors we add redundant states to A (i.e. A_rd0, A_rd1 and A_rd2) and B (i.e. B_rd0, B_rd1 and B_rd2) with the same input, output and same next state. For each row in the original

Table 3.1: FSM benchmark circuits steady state probability: bbara, bbsse, dk14, lion9, shiftreg, train11, cse, keyb and s1.

states	bbara		states	bbsse		states	dk14	
	prob.	accum.		prob.	accum.		prob.	accum.
S2	2.67E-01	2.67E-01	S12	4.50E-01	4.50E-01	S3	2.43E-01	2.43E-01
S5	1.97E-01	4.64E-01	S1	2.25E-01	6.75E-01	S1	1.88E-01	4.31E-01
S1	1.55E-01	6.19E-01	S2	2.14E-01	8.89E-01	S2	1.88E-01	6.19E-01
S3	1.33E-01	7.52E-01	S13	7.50E-02	9.64E-01	S5	1.87E-01	8.06E-01
S4	1.33E-01	8.85E-01	S5	1.45E-02	9.79E-01	S4	1.25E-01	9.31E-01
S6	4.92E-02	9.34E-01	S3	1.38E-02	9.92E-01	S6	5.38E-02	9.85E-01
S8	3.74E-02	9.72E-01	S6	3.63E-03	9.96E-01	S7	1.57E-02	1.00E+00
S7	1.64E-02	9.88E-01	S4	3.46E-03	9.99E-01			
S9	9.36E-03	9.97E-01	S7	2.59E-04	1.00E+00			
s10	2.34E-03	1.00E+00	S8	3.24E-05	1.00E+00			
			S9	9.25E-06	1.00E+00			
			s10	1.18E-06	1.00E+00			
			S11	3.14E-07	1.00E+00			

states	lion9		states	shiftreg		states	train11	
	prob.	accum.		prob.	accum.		prob.	accum.
S1	1.11E-01	1.11E-01	S1	1.25E-01	1.25E-01	S1	1.67E-01	1.67E-01
S2	1.11E-01	2.22E-01	S2	1.25E-01	2.50E-01	S2	8.33E-02	2.50E-01
S3	1.11E-01	3.33E-01	S3	1.25E-01	3.75E-01	S3	8.33E-02	3.34E-01
S4	1.11E-01	4.44E-01	S4	1.25E-01	5.00E-01	S4	8.33E-02	4.17E-01
S5	1.11E-01	5.55E-01	S5	1.25E-01	6.25E-01	S5	8.33E-02	5.00E-01
S6	1.11E-01	6.66E-01	S6	1.25E-01	7.50E-01	S6	8.33E-02	5.84E-01
S7	1.11E-01	7.77E-01	S7	1.25E-01	8.75E-01	S7	8.33E-02	6.67E-01
S8	1.11E-01	8.88E-01	S8	1.25E-01	1.00E+00	S8	8.33E-02	7.50E-01
S9	1.11E-01	9.99E-01				S9	8.33E-02	8.33E-01
						s10	8.33E-02	9.17E-01
						S11	8.33E-02	1.00E+00

states	cse		states	keyb		states	s1	
	prob.	accum.		prob.	accum.		prob.	accum.
S1	8.65E-01	8.65E-01	S1	7.22E-01	7.22E-01	S18	1.20E-01	1.20E-01
S7	3.38E-02	8.99E-01	S3	1.35E-01	8.57E-01	S8	1.16E-01	2.36E-01
s10	3.19E-02	9.31E-01	S4	9.02E-02	9.47E-01	S14	1.14E-01	3.50E-01
S9	2.98E-02	9.61E-01	S2	4.51E-02	9.92E-01	S7	1.05E-01	4.55E-01
S2	2.89E-02	9.89E-01	S6	6.34E-03	9.99E-01	S9	8.55E-02	5.41E-01
S8	6.51E-03	9.96E-01	S7	7.05E-04	9.99E-01	S12	7.71E-02	6.18E-01
S11	2.13E-03	9.98E-01	S5	3.52E-04	1.00E+00	S2	7.06E-02	6.88E-01
S3	1.86E-03	1.00E+00	S9	1.76E-04	1.00E+00	s10	6.61E-02	7.54E-01
S12	1.42E-04	1.00E+00	s10	1.10E-05	1.00E+00	S1	6.04E-02	8.15E-01
S6	6.67E-05	1.00E+00	S12	7.74E-06	1.00E+00	S15	3.12E-02	8.46E-01
S4	6.21E-05	1.00E+00	S8	5.51E-06	1.00E+00	S16	2.75E-02	8.73E-01
S13	2.03E-05	1.00E+00	S15	1.12E-06	1.00E+00	S17	1.93E-02	8.93E-01
S5	4.14E-06	1.00E+00	S13	3.44E-07	1.00E+00	S19	1.92E-02	9.12E-01
S14	1.58E-06	1.00E+00	S17	2.90E-07	1.00E+00	S4	1.69E-02	9.29E-01
S15	6.60E-08	1.00E+00	S11	1.72E-07	1.00E+00	S6	1.47E-02	9.44E-01
S16	5.11E-08	1.00E+00	S19	3.51E-08	1.00E+00	S20	1.27E-02	9.56E-01
			S14	2.15E-08	1.00E+00	S13	1.25E-02	9.69E-01
			S16	5.38E-09	1.00E+00	S11	1.09E-02	9.80E-01
			S18	2.69E-09	1.00E+00	S3	1.05E-02	9.90E-01
						S5	1.02E-02	1.00E+00

Table 3.2: FSM benchmark circuits steady state probability: planet, pma, s832 and s1494.

	planet			pma			s832			s1494	
states	prob.	accum.	states	prob.	accum.	states	prob.	accum.	states	prob.	accum.
S2	5.39E-02	5.39E-02	S7	1.52E-01	1.52E-01	S1	6.40E-01	6.40E-01	S1	8.10E-01	8.10E-01
S7	4.98E-02	1.04E-01	S3	1.37E-01	2.89E-01	S17	2.13E-01	8.53E-01	S15	1.01E-01	9.11E-01
S16	4.98E-02	1.54E-01	S1	1.14E-01	4.03E-01	S18	1.07E-01	9.60E-01	S25	3.80E-02	9.49E-01
S17	4.98E-02	2.03E-01	S4	1.14E-01	5.17E-01	S15	2.00E-02	9.80E-01	S21	1.66E-02	9.66E-01
S18	4.98E-02	2.53E-01	S6	9.49E-02	6.12E-01	S16	1.00E-02	9.90E-01	S17	1.27E-02	9.78E-01
S3	4.15E-02	2.95E-01	S2	9.11E-02	7.03E-01	S3	4.83E-03	9.95E-01	S20	8.31E-03	9.87E-01
S8	4.05E-02	3.35E-01	S11	8.69E-02	7.90E-01	S2	3.79E-03	9.99E-01	S5	4.31E-03	9.91E-01
S9	4.05E-02	3.76E-01	S5	5.69E-02	8.47E-01	S5	7.32E-04	9.99E-01	S34	2.53E-03	9.93E-01
S26	3.73E-02	4.13E-01	S8	3.80E-02	8.85E-01	S4	6.44E-04	1.00E+00	S24	1.75E-03	9.95E-01
S27	3.73E-02	4.50E-01	S21	2.85E-02	9.13E-01	S6	3.05E-06	1.00E+00	S8	1.26E-03	9.96E-01
S4	3.58E-02	4.86E-01	S22	2.85E-02	9.42E-01	S7	2.03E-07	1.00E+00	S13	8.76E-04	9.97E-01
S5	3.13E-02	5.17E-01	s10	2.28E-02	9.65E-01	S19	8.48E-09	1.00E+00	S44	6.32E-04	9.98E-01
S6	3.13E-02	5.49E-01	S9	1.52E-02	9.80E-01	S12	3.69E-09	1.00E+00	S16	3.16E-04	9.98E-01
s10	2.96E-02	5.78E-01	S12	1.45E-02	9.94E-01	S8	1.70E-09	1.00E+00	S36	2.88E-04	9.99E-01
S12	2.96E-02	6.08E-01	S23	3.56E-03	9.98E-01	S24	1.70E-09	1.00E+00	S18	2.19E-04	9.99E-01
S14	2.96E-02	6.37E-01	S13	1.81E-03	1.00E+00	S13	4.69E-10	1.00E+00	S28	2.19E-04	9.99E-01
S19	2.80E-02	6.65E-01	S14	4.52E-04	1.00E+00	S25	1.39E-10	1.00E+00	S23	1.49E-04	9.99E-01
S11	2.02E-02	6.86E-01	S15	4.52E-04	1.00E+00	S14	1.25E-10	1.00E+00	S39	1.09E-04	9.99E-01
S13	2.02E-02	7.06E-01	S17	2.37E-04	1.00E+00	S20	6.02E-11	1.00E+00	S42	1.09E-04	9.99E-01
S15	2.02E-02	7.26E-01	S24	2.22E-04	1.00E+00	S9	5.65E-11	1.00E+00	S30	5.47E-05	9.99E-01
S31	1.87E-02	7.45E-01	S18	3.16E-05	1.00E+00	S23	9.25E-12	1.00E+00	S32	5.47E-05	9.99E-01
S32	1.87E-02	7.63E-01	S16	1.48E-05	1.00E+00	s10	3.77E-12	1.00E+00	S47	5.47E-05	1.00E+00
S43	1.85E-02	7.82E-01	S19	1.98E-06	1.00E+00	S22	6.17E-13	1.00E+00	S38	3.72E-05	1.00E+00
S44	1.85E-02	8.00E-01	S20	1.24E-07	1.00E+00	S21	2.69E-13	1.00E+00	S45	2.79E-05	1.00E+00
S45	1.85E-02	8.19E-01				S11	2.51E-13	1.00E+00	S22	2.74E-05	1.00E+00
S46	1.85E-02	8.37E-01							S2	1.37E-05	1.00E+00
S20	1.49E-02	8.52E-01							S4	3.85E-06	1.00E+00
S28	1.40E-02	8.66E-01							S27	1.38E-06	1.00E+00
S1	1.35E-02	8.80E-01							S14	1.29E-06	1.00E+00
S25	1.12E-02	8.91E-01							S33	1.04E-06	1.00E+00
S33	1.07E-02	9.02E-01							S6	4.89E-07	1.00E+00
S29	9.34E-03	9.11E-01							s10	3.45E-07	1.00E+00
S30	9.34E-03	9.20E-01							S41	2.60E-07	1.00E+00
S36	9.34E-03	9.30E-01							S31	1.97E-07	1.00E+00
S37	9.34E-03	9.39E-01							S19	1.30E-07	1.00E+00
S38	9.34E-03	9.48E-01							S46	1.30E-07	1.00E+00
S42	8.71E-03	9.57E-01							S40	1.22E-07	1.00E+00
S21	7.47E-03	9.65E-01							S29	8.62E-08	1.00E+00
S47	6.22E-03	9.71E-01							S7	6.51E-08	1.00E+00
S34	5.34E-03	9.76E-01							S3	3.70E-08	1.00E+00
S39	4.98E-03	9.81E-01							S11	3.36E-08	1.00E+00
S35	4.00E-03	9.85E-01							S35	2.16E-08	1.00E+00
S22	3.73E-03	9.89E-01							S43	2.16E-08	1.00E+00
S48	3.11E-03	9.92E-01							S37	9.97E-09	1.00E+00
S41	2.49E-03	9.94E-01							S48	8.40E-09	1.00E+00
S23	1.87E-03	9.96E-01							S12	4.99E-09	1.00E+00
S24	1.87E-03	9.98E-01							S9	3.15E-09	1.00E+00
S40	1.87E-03	1.00E+00							S26	2.49E-09	1.00E+00

Table 3.3: FSM benchmark circuits steady state probability: sand, styr and tbk.

	sand			styr			tbk	
states	prob.	accum.	states	prob.	accum.	states	prob.	accum.
S1	1.04E-01	1.04E-01	S1	6.38E-01	6.38E-01	S1	3.42E-01	3.42E-01
S6	9.90E-02	2.03E-01	S13	1.66E-01	8.04E-01	S17	3.42E-01	6.84E-01
S9	7.01E-02	2.73E-01	S11	9.45E-02	8.99E-01	S14	6.56E-02	7.50E-01
S32	6.19E-02	3.35E-01	S12	2.10E-02	9.20E-01	S30	6.56E-02	8.15E-01
S30	5.57E-02	3.91E-01	S2	2.04E-02	9.40E-01	S15	1.16E-02	8.27E-01
S31	5.57E-02	4.46E-01	s10	1.60E-02	9.56E-01	S16	1.16E-02	8.38E-01
S28	4.95E-02	4.96E-01	S3	1.45E-02	9.70E-01	S31	1.16E-02	8.50E-01
S29	4.95E-02	5.45E-01	S14	1.28E-02	9.83E-01	S32	1.16E-02	8.62E-01
S8	4.38E-02	5.89E-01	S9	1.07E-02	9.94E-01	S2	5.79E-03	8.67E-01
S26	4.33E-02	6.33E-01	S15	3.49E-03	9.97E-01	S3	5.79E-03	8.73E-01
S27	4.33E-02	6.76E-01	S4	6.60E-04	9.98E-01	S4	5.79E-03	8.79E-01
S24	3.71E-02	7.13E-01	S29	3.47E-04	9.98E-01	S5	5.79E-03	8.85E-01
S25	3.71E-02	7.50E-01	S8	3.30E-04	9.99E-01	S6	5.79E-03	8.91E-01
S22	3.09E-02	7.81E-01	S16	2.18E-04	9.99E-01	S7	5.79E-03	8.96E-01
S23	3.09E-02	8.12E-01	S17	1.96E-04	9.99E-01	S8	5.79E-03	9.02E-01
S20	2.48E-02	8.37E-01	S23	1.25E-04	9.99E-01	S9	5.79E-03	9.08E-01
S21	2.48E-02	8.61E-01	S7	3.07E-05	9.99E-01	s10	5.79E-03	9.14E-01
S18	1.86E-02	8.80E-01	S5	2.16E-05	9.99E-01	S11	5.79E-03	9.20E-01
S19	1.86E-02	8.99E-01	S6	2.06E-05	9.99E-01	S12	5.79E-03	9.25E-01
S2	1.84E-02	9.17E-01	S18	1.96E-05	9.99E-01	S13	5.79E-03	9.31E-01
S7	1.65E-02	9.34E-01	S30	1.64E-05	9.99E-01	S18	5.79E-03	9.37E-01
S16	1.24E-02	9.46E-01	S20	1.36E-05	9.99E-01	S19	5.79E-03	9.43E-01
S17	1.24E-02	9.58E-01	S24	4.31E-06	9.99E-01	S20	5.79E-03	9.48E-01
S11	8.25E-03	9.67E-01	S26	4.04E-06	9.99E-01	S21	5.79E-03	9.54E-01
S13	8.25E-03	9.75E-01	S22	2.18E-06	9.99E-01	S22	5.79E-03	9.60E-01
S14	6.19E-03	9.81E-01	S21	1.36E-06	9.99E-01	S23	5.79E-03	9.66E-01
S15	6.19E-03	9.87E-01	S19	1.23E-06	9.99E-01	S24	5.79E-03	9.72E-01
S3	4.80E-03	9.92E-01	S28	1.54E-07	9.99E-01	S25	5.79E-03	9.77E-01
s10	4.38E-03	9.96E-01	S27	1.39E-07	9.99E-01	S26	5.79E-03	9.83E-01
S12	2.06E-03	9.98E-01	S25	1.35E-07	9.99E-01	S27	5.79E-03	9.89E-01
S4	7.07E-04	9.99E-01				S28	5.79E-03	9.95E-01
S5	7.07E-04	1.00E+00				S29	5.79E-03	1.00E+00

Table 3.4: State table of the bit flipper example before and after protection.

Original state table				Protected state table			
I	PS	NS	O	I	PS	NS	O
0	A (0)	A (0)	0	0	A (000)	A (000)	0
				0	A_rd0 (100)	A (000)	0
				0	A_rd1 (010)	A (000)	0
				0	A_rd2 (001)	A (000)	0
0	B (1)	B (1)	1	0	B (111)	B (111)	1
				0	B_rd0 (011)	B (111)	1
				0	B_rd1 (101)	B (111)	1
				0	B_rd2 (110)	B (111)	1
1	A (0)	B (1)	0	1	A (000)	B (111)	0
				1	A_rd0 (100)	B (111)	0
				1	A_rd1 (010)	B (111)	0
				1	A_rd2 (001)	B (111)	0
1	B (1)	A (0)	1	1	B (111)	A (000)	1
				1	B_rd0 (011)	A (000)	1
				1	B_rd1 (101)	A (000)	1
				1	B_rd2 (110)	A (000)	1

state table, additional redundant rows are added in the protected state table as shown in Table 3.4. The number of the additional rows is equal to the number of redundant states times the number of rows in the original state table.

In order to detect all single errors, the hamming distance between state A, and its redundant states should be 1. Similarly for state B, the hamming distance between it and any of its redundant states should be 1. Therefore, the distance between states A and B must be kept at least 3. Errors can happen either when the FSM is in a state (i.e. it happens in one of the FFs) or going to one of them (i.e. it happens in the combinational logic). The distance-3 relation is necessary so that any single bit error that occurs while being in or during transition to state A or B will lead us to equivalent redundant state. Figure 3.2 shows the final

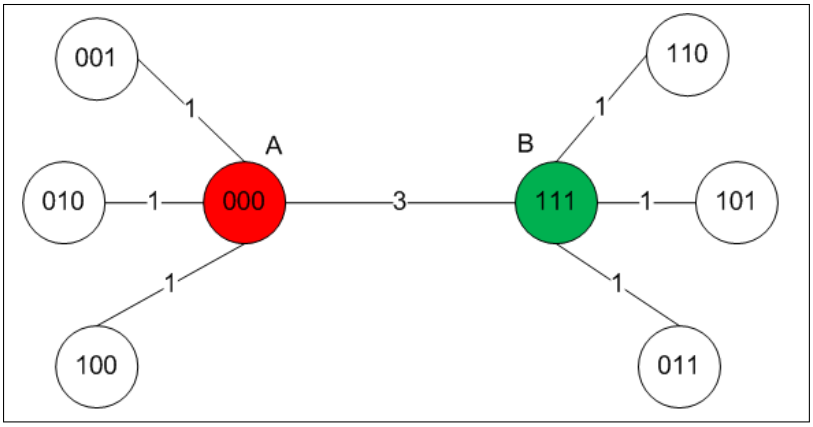


Figure 3.2: State diagram and encoding with two protected states.

state diagram and encoding after adding the redundant states. In Figure 3.2, the minimum possible number of bits to represent all protected and redundant states is 3 bits. This is why we need to add 3 redundant states to cover all single errors in the state encoding bits. Eventually, we need 3 D-FFs to represent this circuit. If we use less than 3 D-FFs, this will result into 2 or more redundant states having the same encoding which is not allowed.

Moreover, SET faults occurring in the combinational logic are protected by not sharing logic in the combinational logic blocks implementing the next state equations of memory elements or flip flops. This is done by partitioning the combinational logic in the input cones of each D-FF so that no single error can propagate to more than one D-FF. The final block diagram after adding the redundant states should be as depicted in Figure 3.3.

If soft error tolerance for all states is considered, this might result in excessive area, delay and power overhead. Fortunately, as mentioned in Section 3.2 and shown in Tables 3.1, 3.2 and 3.3, many sequential circuits have few states

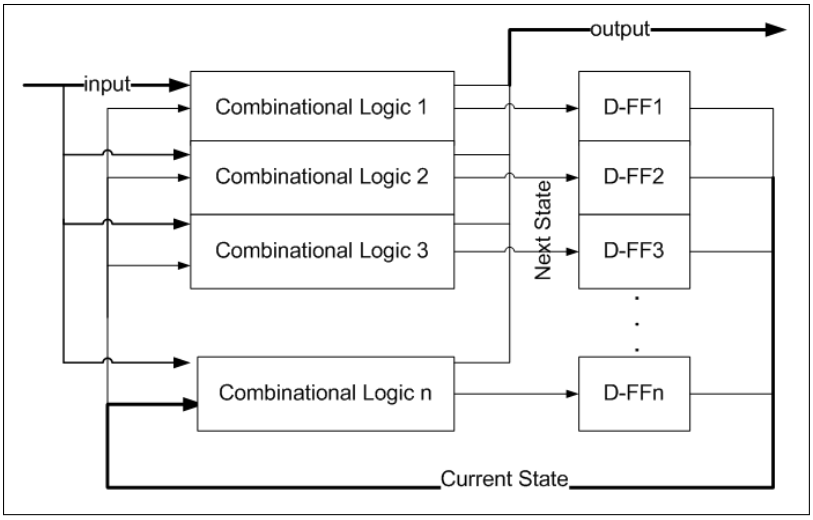


Figure 3.3: Sequential circuits block diagram with redundant states.

Table 3.5: Steady state probability of benchmark circuit bbsse.

State	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13
Prop.	2.25 E-1	2.24 E-1	1.38 E-2	3.46 E-3	1.45 E-2	3.63 E-3	2.59 E-4	3.24 E-5	9.25 E-6	1.18 E-6	3.14 E-7	4.50 E-1	7.50 E-2

with high probability of occurrence. For example, Table 3.5 shows the steady state probabilities of state occurrence for benchmark circuit bbsse. As shown, the probability of state occurrence varies significantly from state to state. Only 3 states out of 13, namely S12, S1 and S2, have relatively high probability of occurrence and may be considered for reliability enhancement. Taking advantage of this and enhancing the reliability of states with the highest probability of occurrence only, the overall area overhead of the circuit is minimized.

3.3.2 Calculating the Minimum Number of Bits Used in State Encoding

The state assignment problem of an FSM can be viewed as a coding problem or as a partitioning problem [69–72]. The coding problem requires each state to be assigned a unique binary pattern. Determining the minimum number of bits required in state encoding after adding the redundant states needs further investigation. For example, suppose we have only 2 protected states, the number of bits (or D-FFs) used is 3 bits. With 3 bits we can represent $2^3 = 8$ states. In Figure 3.4, since we cover all single errors, the hamming distance between the protected states and the redundant state must be 1. Moreover, the distance between the redundant states and any other protected state must be more than 1.

In order to specify the number of extra bits needed to encode all states (original and redundant), we will compute it based on the total number of states. Let m be the number of the normal states in FSM. Let n be the number of protected states, and α be the number of bits needed to encode the protected states. Therefore, $\alpha = log_2(n)$. We want to compute the number of extra bits needed, β, in order to encode all the states.

The total number of bits used in encoding is $\alpha+\beta$, which can be used to encode $2^{\alpha+\beta}$ states. Each protected state will have $\alpha+\beta$ redundant states, since all single-bit errors are considered (i.e. all the distance 1 encodings are redundant to each protected state). Therefore, the total number of states will be $n(1 + \alpha + \beta) + m$.

The condition for the number of encoding bits needed is that the number of states must be less than or equal to the number of possible encodings. Therefore, we get $2^{\alpha+\beta} \geq n(1+\alpha+\beta)+m$. This inequality can be used to find the minimum number of bits needed in order to add redundancy to states.

However, we find that this inequality works only with values of $n = 2^i$ assuming that $m = 0$. This means that if the number of states is not a power of 2, more conditions must be taken into account. The only condition that we can add is to ensure that there is no overlapping between redundant states.

To illustrate this, let us assume that 3 states need to be protected with no normal states. According to the above discussion ($2^2 * 2^2 \geq 3(1+2+2)$). Thus, we use 4-bits to encode them. However, this will result in overlapping between some of the redundant states. Figure 3.4 shows that states A and C have the same redundant states namely, 0100 and 1000. Similarly, states B and C have the same redundant states namely, 1101 and 1110. The correct encoding is to use 5-bits instead of 4-bits. One possible encoding is shown in Figure 3.5.

The hamming distance between states must be calculated carefully to get the minimum possible number of bits to design our system. To ensure that the redundant states do not overlap, the minimum possible hamming distance between the protected states must be at least 3. The reason behind this is to make the distance between any two different redundant states greater than 0. Table 3.6 gives one possible encoding for a given number of states to be protected.

Notice that in Table 3.6, every state has a hamming distance of 3 or more

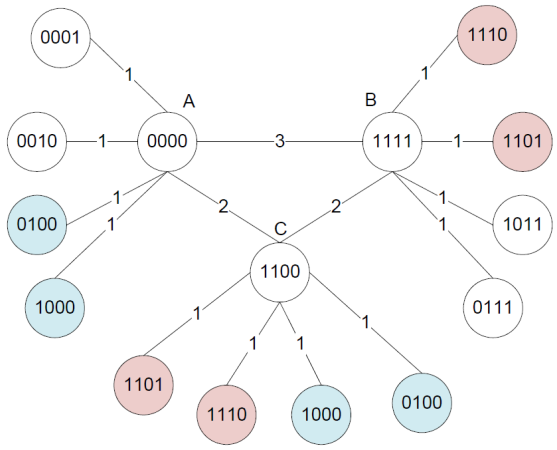


Figure 3.4: Incorrect encoding for 3 states with redundancy.

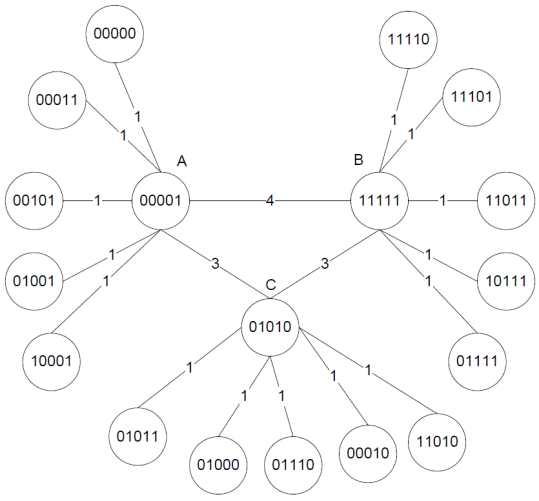


Figure 3.5: Correct encoding for 3 states with redundancy.

Table 3.6: A possible encoding for different number of protected states.

Number of protected states.	Number of required bits.			one possible encoding	Remaining codes for normal states.
	α	B	α +β		
2	1	2	3	{(000),(111)}	2^3 -(2*(3+1)) = 0
3	2	3	5	{(0_0000),(0_0111),(1_1001)}	2^5 -(3*(5+1)) = 14
4	2	3	5	{(0_0000),(0_0111),(1_1001),(1_1110)}	2^5 -(4*(5+1)) = 8
5	3	3	6	{(000_000),(000_111),(011_001),(011_110),(101_010)}	2^6 -(5*(6+1)) = 29
6	3	3	6	{(000_000),(000_111),(011_001),(011_110),(101_010), (101_101)}	2^6 -(6*(6+1)) = 22
7	3	3	6	{(000_000),(000_111),(011_001),(011_110),(101_010), (101_101),(110_011)}	2^6 -(7*(6+1)) = 15
8	3	3	6	{(000_000),(000_111),(011_001),(011_110),(101_010), (101_101),(110_011),(110_100)}	2^6 -(8*(6+1)) = 8
9	4	3	7	{(000_0000),(0000_0111),(1100_001),(0011_110), (1100_110),(1010_100),(0011_001),(1111_000),(111_1111)}	2^7 -(9*(7+1)) = 56
10	4	3	7	{(000_0000),(0000_0111),(1100_001),(0011_110), (1100_110),(0101_100),(1010_100),(0011_001),(1111_000),(111_1111)}	2^7 -(10*(7+1)) = 48
11	4	3	7	{(000_0000),(0000_0111),(1100_001),(0011_110), (1100_110),(0101_100),(1010_011),(1010_100),(0011_001),(1111_000),(111_1111)}	2^7 -(11*(7+1)) = 40
12	4	3	7	{(000_0000),(0000_0111),(1100_001),(0011_110), (1100_110),(0101_100),(1010_011),(0101_011),(1010_100),(0011_001),(1111_000),(111_1111)}	2^7 -(12*(7+1)) = 32
13	4	3	7	{(000_0000),(0000_0111),(1100_001),(0011_110), (1100_110),(0101_100),(1001_101),(1010_011),(0101_011),(1010_100),(0011_001),(1111_000),(111_1111)}	2^7 -(13*(7+1)) = 24
14	4	3	7	{(000_0000),(0000_0111),(1100_001),(0011_110), (1100_110),(0101_100),(1001_101),(0110_101),(1010_011),(0101_011),(1010_100),(0011_001),(1111_000),(111_1111)}	2^7 -(14*(7+1)) = 16
15	4	3	7	{(000_0000),(0000_0111),(1100_001),(0011_110), (1100_110),(0101_100),(1001_101),(1001_010),(0110_101),(1010_011),(0101_011),(1010_100),(0011_001),(1111_000),(111_1111)}	2^7 -(15*(7+1)) = 8
16	4	3	7	{(000_0000),(0000_0111),(1100_001),(0011_110), (1100_110),(0101_100),(1001_101),(1001_010),(0110_010),(0110_101),(1010_011),(0101_011),(1010_100),(0011_001),(1111_000),(111_1111)}	2^7 -(16*(7+1)) = 0

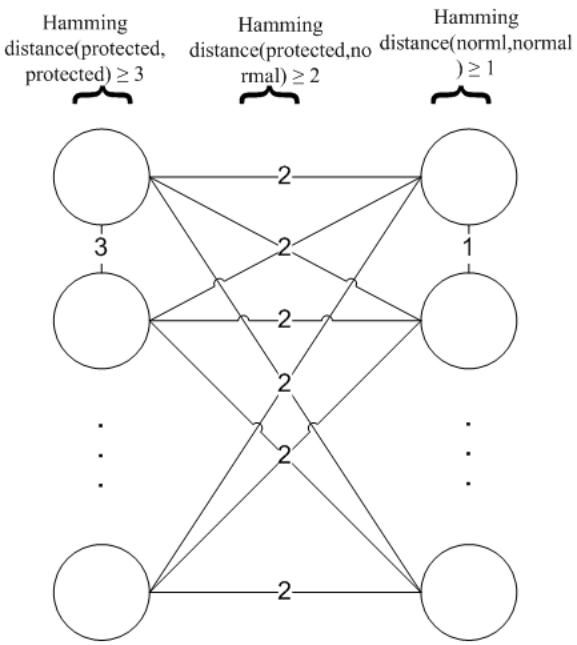


Figure 3.6: Hamming distance between different types of states.

from all other protected states. It must also be ensured that every other state (protected and redundant) has a hamming distance of at least 2 with each normal state. Figure 3.6 shows the hamming distance relation between different types of states.

Lower Bound and Upper bound on Number of Encoded Bits

If we assume that normal states, m = 0, we want to find the lower bound and the upper bound that enclose the number of bits used to encode the states in a sequential circuit after protecting the most probable states. Given that: n: number of protected states; m: number of normal states, α : number of bit needed to encode all protected states; β: extra bits needed to encode protected states and their redundant states; $\alpha + \beta$: total number of bits used to represent

FSM; $n(1+\alpha+\beta)$: number of protected states and their redundant states. Let us assume $m = 0$. If n and α are given, we need to find β such that:

$$2^{\alpha+\beta} \geq n(1+\alpha+\beta)$$

and ,

hamming distance between all the protected states is 3.

This en-quality serves as a lower bound, since we have to cover all the states (i.e. protected and redundant), while the hamming distance ensures that there is no overlapping between the different types of states.

The upper bound is $2^{\alpha+\alpha+\alpha} = 2^{3\alpha}$, because the hamming distance between the codes of states in the original FSM is 1 and triplicating their codes guarantees a hamming distance of 3.

If we combine both the lower bound and the upper bound, the resulting inquality is: If n and α are given, we need to find β such that:

$$2^{3\alpha} \geq 2^{\alpha+\beta} \geq n(1+\alpha+\beta)$$

or

$$3\alpha \geq \alpha+\beta \geq log_2(n(1+\alpha+\beta))$$

and ,

hamming distance between all the protected states is 3.

In order to illustrate this, let us consider some examples. Assume $n = 2$ and $\alpha = 1$, we need to find β such that: $3 \geq 1+\beta \geq log_2(2(2+\beta))$. Trying $\beta = 1$, the

condition is not satisfied because: $3 \geq 2 \geq log_2(6)$ is violating the lower bound. Trying $\beta = 2$, the condition is satisfied because: $3 \geq 3 \geq 3$ is a correct statement. Therefore, the only choice we have is to use 3 bits. One chosen codes are: 000 and 111.

Considering another example, assume $n = 3$ and $\alpha = 2$, we need to find β such that: $6 \geq 2+\beta \geq log_2(3(3+\beta))$. Trying $\beta = 1$, the condition is not satisfied because: $6 \geq 3 \geq log_2(12) = 3.59$ is violating the lower bound. Trying $\beta = 2$, the condition is satisfied because: $6 \geq 4 \geq log_2(15) = 3.91$ is a correct statement. Trying $\beta = 3$, the condition is satisfied because: $6 \geq 5 \geq log_2(18) = 4.17$ is a correct statement. Trying $\beta = 4$, the condition is satisfied because: $6 \geq 6 \geq log_2(21) = 4.39$ is a correct statement. We can choose 4 or 5 or 6 bits, but we must ensure that the hamming distance between the codes is 3 at least. If we use 4-bits to encode them; this will result in overlapping between the redundant states as shown in Figure 3.4. The correct encoding is to use 5-bits instead of 4-bits as shown in Figure 3.5.

Figure 3.7 plots the lower bound and upper bound as well as the actual minimum number of bits used to encode a given number of protected states. Assuming no normal states, m = 0, the X-axis represents the protected states and Y-axis represents the number of bits used. As illustrated in the figure, the lower bound is very tight. Therefore, in order to search for the minimum number of bits used to encode the protected states, it is better to start from the lower bound and increase the number of bits until the hamming distance constraint is satisfied. Table 3.7

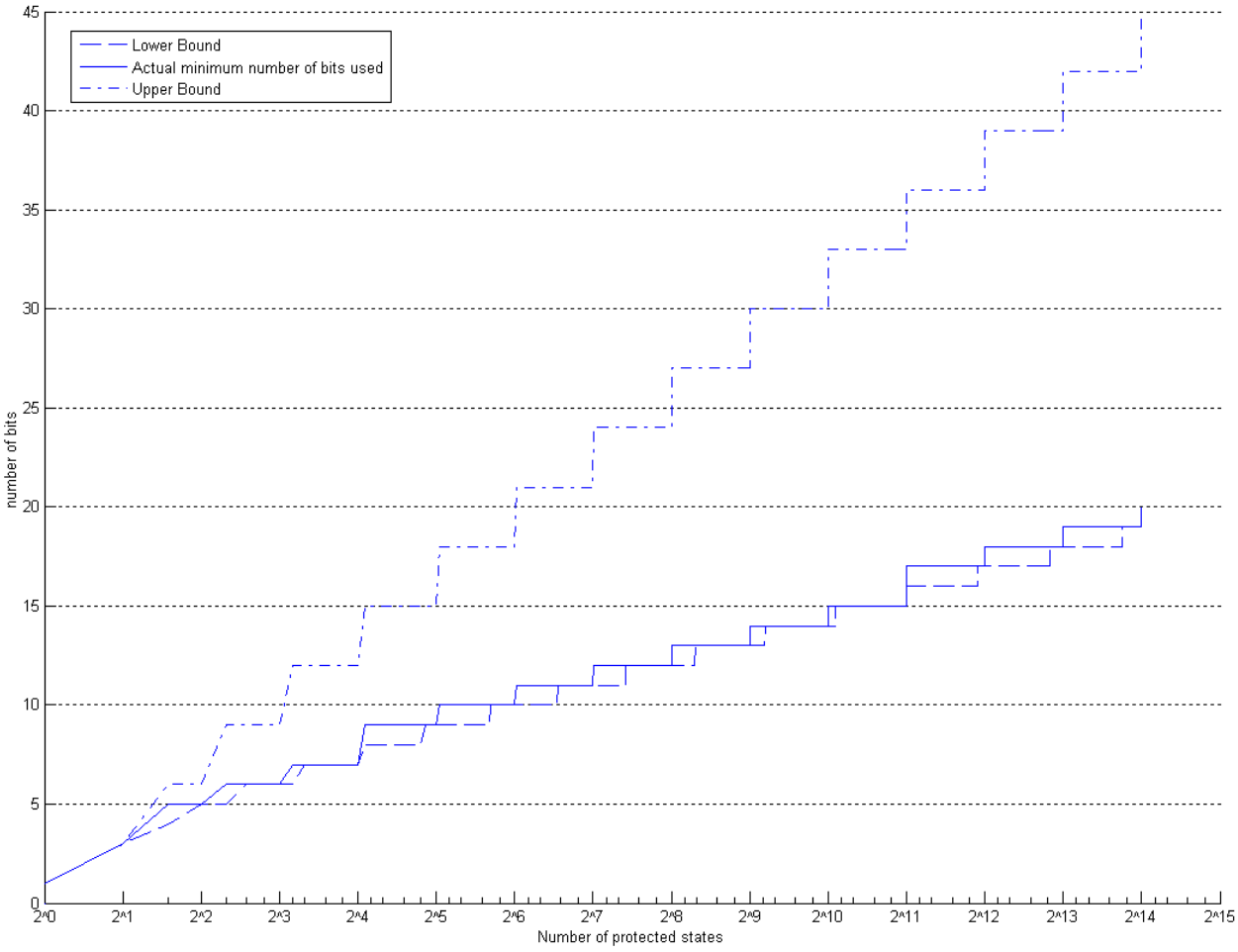


Figure 3.7: Lower bound, actual minimum number of bits used, and upper bound.

shows selected number of protected states, their lower bound, actual minimum number of bits used to encode them, and upper bound. At each $2^{\alpha}+1$, the minimum number of bits used to encode the protected states in hand is incremented either by 1 bit as in 2^2+1, 2^3+1, 2^5+1, 2^6+1, 2^7+1, 2^8+1, 2^9+1, $2^{10}+1$, $2^{12}+1$, $2^{13}+1$ and $2^{14}+1$; or by 2 bits as in 2^1+1, 2^4+1 and $2^{11}+1$.

Noting that the hamming distance condition is more restrictive than the lower bound condition, we will focus in trying to get hamming distance 3 between the codes starting from the lower bound as it will be discussed in the next sub-section.

Table 3.7: Lower bound, actual minimum number of bits used, and upper bound for a selected number of protected states.

Power of 2	Number of protected states	Lower bound	Minimum number of bit used	Upper bound
2^0	1	1	1	1
2^1	2	3	3	3
	3	4	5	3
2^2	4	5	5	6
	5	5	6	6
	6	6	6	6
2^3	8	6	6	9
	9	6	7	9
	10	7	7	9
2^4	16	7	7	12
	17	8	9	12
	28	8	9	12
	29	9	9	12
2^5	32	9	9	15
	33	9	10	15
	51	9	10	15
	52	10	10	15
2^6	64	10	10	18
	65	10	11	18
	93	10	11	18
	94	11	11	18
2^7	128	11	11	21
	129	11	12	21
	170	11	12	21
	171	12	12	21
2^8	256	12	12	24
	257	12	13	24
	315	12	13	24
	316	13	13	24
2^9	512	13	13	27
	513	13	14	27
	585	13	14	27
	586	14	14	27
2^{10}	1024	14	14	30
	1025	14	15	30
	1092	14	15	30
	1093	15	15	30
2^{11}	2048	15	15	33
	2049	16	17	33
	3855	16	17	33
	3856	17	17	33
2^{12}	4096	17	17	36
	4097	17	18	36
	7281	17	18	36
	7282	18	18	36
2^{13}	8192	18	18	39
	8193	18	19	39
	13797	18	19	39
	13798	19	19	39
2^{14}	16385	19	19	42
	16386	19	20	42

3.3.3 Algorithm 1: Minimum State Redundancy Algorithm for Fault Tolerance

Since we do not have exact formula for the minimum number of bits used to encode states in a protected FSM, Algorithm 1 shown in Tables 3.8, 3.9 and 3.10 uses exhaustive search to find codes for protected, redundant and normal states. The idea is based on protecting few high probable states. Therefore, it is highly unlikely to have protected states beyond 100 states. Hence, although exhaustive search methods consume a lot of time to find the solution, in our case it is acceptable.

Algorithm 1 can be divided into few portions, namely, *getting the lower bound*, *getting the protected states codes*, *checking if the normal states are covered by the remaining codes based on the current number of bits used for encoding*, *getting the redundant states codes* and finally *getting the normal states codes*.

In Table 3.8, inputs are the number of protected states and the number of normal states. After finding α and β, we initialize empty sets for the outputs. Then, we find the lower bound based on the formula given in Section 3.3.2. Iterations, starting from the lower bound until the protected states are covered, are inspected by searching for codes with distance 3 between them. In each iteration, the current codes are tested if they result in new codes with distance 3 between them. If there are no new codes resulting form the current codes, the current codes are padded with "0" from the left and the number of bits used so far is incremented by 1. Otherwise, the new codes are added to the current ones. The

outputs, as shown in Table 3.9, are the protected states codes, redundant states codes and normal states codes along with the minimum number of bits used.

Based on the obtained number of bits used to encode the protected states, the remaining codes are investigated whether they cover the required normal states. The remaining codes at this point is basically the possible codes obtained by current number of bits used minus the number of the protected states and their redundant states. This amount is $2^{\alpha+\beta} - n(1+\alpha+\beta)$ where $\alpha+\beta =$ *number of bits used*.

The current number of bits used in the encoding, $\alpha+\beta$, is incremented by γ bits to cover all the normal states, m. The number of extra bits which is needed to cover all normal states, γ, must satisfy that all the possible codes obtained by, $2^{\alpha+\beta+\gamma}$, cover the normal states, m, the protected states, n, and their redundant states, $n(\alpha+\beta+\gamma)$. By solving for x in the following equation:

$$2^{\alpha+\beta+x} = m + n(\alpha+\beta+x+1),$$

the additional number of bits needed to cover all the normal states can be obtained by $\gamma = \lceil x \rceil$.

Another alternative way to find the additional number of bits needed to cover all the normal states, γ, is by incrementing the current number of bits used to encode the protected states and their redundant states, $\alpha+\beta$, by 1 bit and investigated the amount: $2^{\alpha+\beta} - n(1+\alpha+\beta)$ again iteratively until the normal states are covered.

After that, the redundant states codes are computed for each protected state.

They are easily found by flipping each bit of the protected state code to get 1 redundant state code at a time. For example, if the protected states code is "000000", the one-hot-encoding for 6 bits is the codes for its corresponding 6 redundant states [73]. Then, for each normal state, one of the remaining codes is assigned to it. This is shown in Table 3.9. Based on Table 3.6, it is required that each normal state has a hamming distance of 2 between the protected states and itself. This condition is clearly satisfied since all the distance 1 codes from the protected states codes are assigned to the redundant states. Finally, the outputs are reported which are: the protected states codes, the redundant states codes, normal states codes and the minimum number of bits used to encode all of them.

The core idea of the algorithm is based on how we find codes with distance 3 between them. In Table 3.10, the function, *get_distance3_codes()*, is described. Taking the current codes for the protected states, and the number of bits currently used to encode them as arguments, it runs through all the codes obtained from the current bit used for encoding. For each code of the current protected states codes, if the distance between the investigated current code and all the the existing protected states codes is 3 or more, the current code is added to the protected states codes. Other wise, the next code is investigated and so on.

3.3.4 Complexity of Algorithm 1

The complexity of Algorithm 1 is mainly controlled by the applied method to find the distance 3 codes between the protected states. As it is required that

Table 3.8: Algorithm 1.

Inputs	• Get the number of *protected states* and the number of *normal states* as inputs.
Initialize	• Compute α as $\alpha = \lceil log_2(protected\ states) \rceil$ and let $\beta = 1$. • Let the size of *protected state codes* $= 0$.
Lower bound	• Compute the *lower bound* by incrementing β iteratively starting from *lower bound* $= \lceil log_2(protected\ states(1+\alpha+\beta)) \rceil$ until $\alpha+\beta \geq$ *lower bound*.
Protected states	• Assign the current *number of bits used* to the lower bound. • Let the first protected state code be $0 \cdots 00$. Its length is the current *number of bits used*. • Increment the size of the *protected state codes* by 1. • While the size of the *protected state codes* is less than the number of *protected states*: – Get all the hamming distance 3 codes from all of the current *protected state codes* using get_distance3_codes() (see Table 3.10) taking the current *number of bits used* as an argument. – If no new codes are found, the *number of bits used* is incremented by 1 and all the current *protected state codes* are padded with 0 from the left. Otherwise, the new codes are added to the *protected state codes*.
Find γ	• Find the extra number of bits needed to encode all the *normal states*, γ. • Increment the current *number of bits used* by γ.
Redundant states	• For each *protected state*, assign all hamming-distance-1 codes as its *redundant state codes*.

Table 3.9: Algorithm 1 - continued

Normal states	• Codes that are not used for *protected state codes* or *redundant state codes* are assigned to *normal state codes*.
Outputs	• Collect *protected state codes*, *redundant state codes*, *normal state codes* and *number of bits used* as outputs.

Table 3.10: Algorithm 1 - get_distance3_codes.

Get distance-3 codes	Get_distance3_codes() returns a new set of *protected state codes* with distance 3 between them starting from the current *protected state codes* with a certain *number of bits used* in the encoding, as follows: • Run through all bit strings of length equal to the *number of bits used* in the encoding indexed by i or until all *protected states* are covered. – Run through all the *protected state codes* indexed by j. ∗ Calculate the hamming distance between (code i, code j). – If the hamming distance of code i is at least 3 from all the codes indexed by j, include it in the *protected state codes*. • Return the new set of the *protected state codes*.

each code in the set of the protected states codes has distance 3 between them, the search of such codes goes through 3 nested loops. The outer loop is the one that calls the function *get_distance3_codes()*; it costs roughly ($K-$ *lower bound*) where K is the number of bits used to encode the protected state codes. However, since the this a small number of iterations given that the lower bound is tight, it can be neglected. The other 2 loops are inside the function *get_distance3_codes()*. The first one runs from 0 to 2^K and the other one runs from 0 to n where n is the number of protected state codes. These two loops cost $n * 2^K$. Therefore, the complexity of Algorithm 1 is $O(n * 2^K)$. This cost is acceptable since the number of protected states is often small.

3.4 Correlation Between State Probability Based Protection and Failure Rate

In this section, we will explore the relation between the protection of states according to their probability of occurrence and the failure rate. In order to show it, the sequential benchmark circuit dk14 is taken as an example. The FSM of dk14 contains 7 states. Figure 3.8 shows the failure rate of the circuit vs the number of injected faults, obtained by applying Algorithm 1 to dk14. As shown in Table 3.11, the states are sorted in descending order of their probability of occurrence as follows: state-3, state-1, state-2, state-5, state-4, state-6 and state-7. The original circuit dk14 without adding redundancy has the worst failure rate.

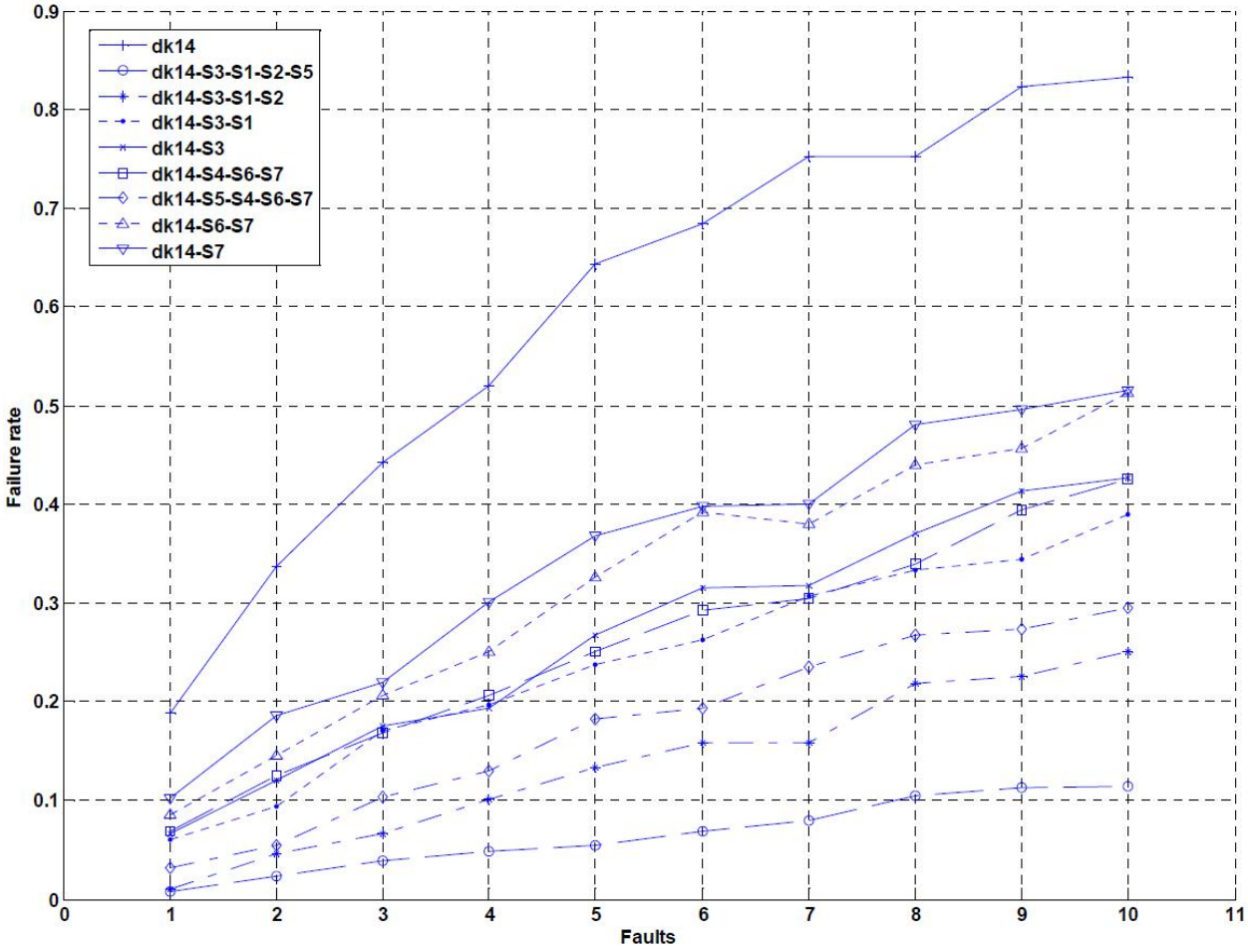


Figure 3.8: Correlation between state probability based protection and failure rate for benchmark sequential circuit dk14.

Other versions of dk14 have been built based on state protection. Four pairs of experiments were made by protecting the highest probable state versus the lowest one; the two highest probable states versus the two lowest ones; the three highest probable states versus the three lowest ones; and the four highest probable states versus the four lowest ones.

Figure 3.8 shows the following relations:

- The failure rate decreases as more states with higher probability of occur-

Table 3.11: Steady state probability for dk14.

states	dk14	
	prob.	accum.
S3	2.43E-01	2.43E-01
S1	1.88E-01	4.31E-01
S2	1.88E-01	6.19E-01
S5	1.87E-01	8.06E-01
S4	1.25E-01	9.31E-01
S6	5.38E-02	9.85E-01
S7	1.57E-02	1.00E+00

rence are protected. We can see that by comparing the protection of highest state, 2 highest states, 3 highest states and 4 highest states.

- Protecting the highest probable states is always better than protecting the lower ones. We can see that by comparing the 4 pairs of experiments, each pair at a time.

- Protecting 1 or few higher probable states with certain probability is better than protecting many states with cumulative probability less than the higher ones. For example, protecting the 2 highest states is better than protecting the 3 lowest ones.

In summary, there is a strong relation between the state probability based protection and the failure rate or fault tolerance. Logically, protecting the highest probable state gives a better fault tolerance than protecting the lowest probable state. In other words, the failure rate of sequential circuits which involves protecting states with higher probability of occurrence is less than the ones involving protecting random or lower state probability.

3.5 Comparing the Proposed Algorithm with Other Techniques

Increasing the reliability of sequential circuits has been studied by many researchers. Solutions have been proposed to reduce SEU and SET by approaching the problem from different perspectives. Most of the popular methods are concerned with the importance of the flip flops in the design. Based on it, protection methods are adopted to harden or TMR them. Knowing that each next state variable corresponds to one flip flop function, selective hardening can be applied. If the designated method protects/Hardens latches against SEU, it might not protect the combinational logic against SET. Although some approaches protect sequential circuit against both SEU and SET, they did not necessary protect against multiple faults, namely SEMU. Moreover, faults that span for more than one cycle might cause incorrect behavior of the circuit.

In our method, we look at the problem from a different perspective. Rather than looking at each next state variable, we measure the importance of the states themselves. The importance of the states is directly related to the state probability which is more realistic and more close to the behavior model of the FSM of sequential circuits. In other words, instead of protecting subset of the Flip flops often presented in the implementation level, we distribute the protection among all the Flip flops by protecting subset of the states.

In our method we protect sequential circuit against both SET and SEU. Moreover, SEMU can be tolerated if they happen in the same flip flip cone, or if after

masking result in a single fault. Also, faults (i.e. SEU or SET) that occur in consecutive cycles are tolerated if a SEU (SET) occurring in one cycle is followed by a SEU (SET) in the next cycle. Because they will cause the sequential circuit machine to be in one of the equivalent redundant states. Also, they are tolerated when a SEU occurring in one cycle is followed by a SET in the next cycle. Because they will cause the sequential circuit machine to be in one of the equivalent redundant states. However, if a SET occurring in one cycle is followed by a SEU in the next cycle, they might cause double fault that cause the sequential circuit machine to be in a different non-equivalent state. To achieve tolerance against these kind of faults, two levels of redundant states are needed.

To show this, let us consider the simple example depicted in Figure 3.2. Two states are protected using level 1 redundant states only which ensures protection against single faults that are spanning for one cycle. Figure 3.9 shows protection of these 2 states using two levels of redundant states. This time protection of faults that are spanning for 2 cycles is guaranteed. This means when a fault happens in the original protected states it leads to one of the level 1 equivalent redundant states. If another fault happens while being in one of the level 1 redundant stats, either it leads to one of the level 2 redundant states or it brings the machine back to the original protected state if the 2^{nd} fault overwrites the first one. For example, if the FSM was in the protected state "00000" and a fault happens in the first bit, it will lead to an equivalent redundant state "00001" which is one of level 1 redundant states. If in the 2^{nd} cycle another fault hit the 3^{rd} bit, for example,

it will lead to "00101" which is another level-2 equivalent redundant state of the state "00000". Moreover, if in the 2^{nd} cycle a fault happens in the first bit again that toggles it from "1" to "0", the FSM returns back to "00000" which is the original protected state.

However, adding a second level of redundant states will produce area overhead much larger than using a first level. Also, it complicates the hamming distance condition between the protected states. For example, In Figure 3.2 we used 3 bits to encode all the states and 3 redundant states are added to each of the two protected states. Hence, the total amount of states is 8 for protecting 2 states. On the other hand, protecting them using 2 level redundant states will cost us 10 redundant states at level 1 and 20 at level 2. The total amount of states will be 32 for protecting 2 states which is encoded using 5 bits. Hence, the area overhead is increased in terms of the number of D-FFs as well as the combinational logic witch are directly affected by the number of states in the FSM. Moreover, the hamming distance between the protected states using two level redundant states should be at least 5 compared to 3 for one level redundant states. Hence, more bits should be added to the original state encoding to ensure that no 2 redundant states have the same encoding.

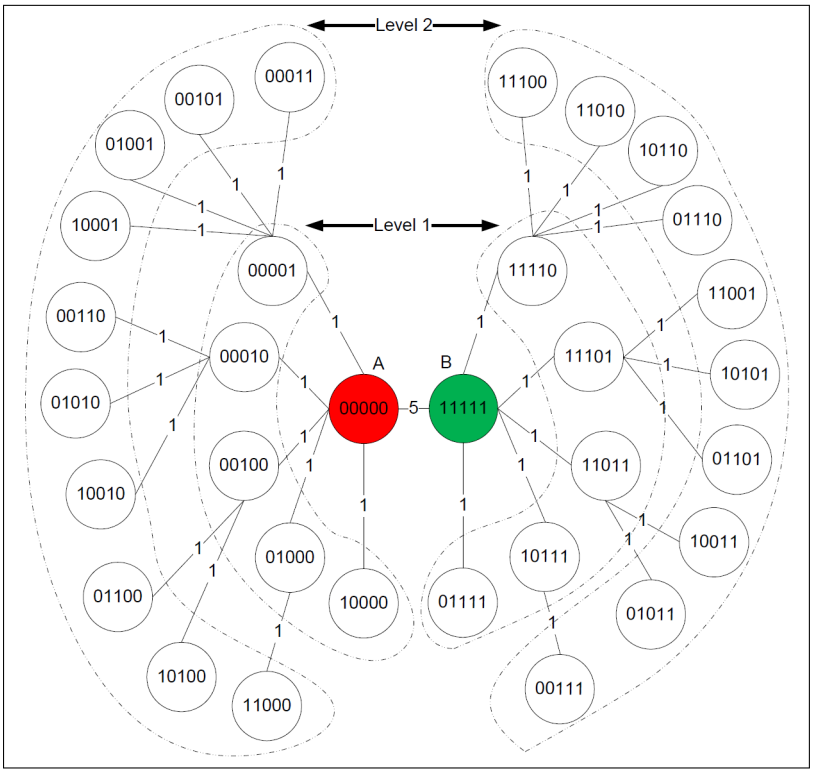


Figure 3.9: State encoding with two protected states using 2 level of redundant states.

3.6 Framework of Enhancing Fault Tolerance Using Algorithm 1

Our method presented in this chapter, gives a procedure that enables protecting highest probable states to achieve tolerance against soft errors. Dynamically, sequential circuits reliability can be increased as more states with high probability of occurrence are protected. The area overhead is also increased as a result. This is the case up to a certain point at which less drop in failure rate is achieved at a higher area overhead price. By adopting this method, the following framework should be applied in the design procedure, as shown in Figure 3.10:

1. Given a sequential circuit represented by a state table or finite state machine (FSM).

2. Calculate the steady state probability of all the states in the FSM as shown in Section 3.2.

3. Protect the chosen high probable states using Algorithm 1 by introducing redundant equivalent states as shown in section 3.3.

4. Synthesize/implement the sequential circuit.

3.7 Conclusion

In this chapter, we have introduced a novel idea, presented as Algorithm 1, to increase sequential circuit reliability and hence fault tolerance by introducing

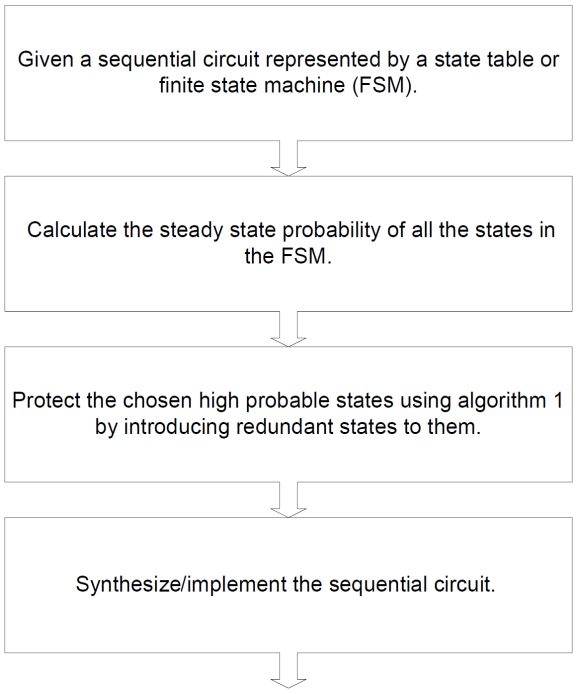


Figure 3.10: Framework of Algorithm 1.

redundant equivalent states to the states with high probability of occurrence in sequential circuit behavioral machine. To maintain the same operation of the unprotected FSM, the newly added redundant states have the same input, next state, and output as the original states. Other states with low probability are kept without adding redundancy to minimize the area overhead.

We divided the original states of sequential circuits into *protected states* with high probability of occurrence and *normal states* with low probability of occurrence. For each protected state, equivalent *redundant states* are added to guarantee single soft fault tolerance. We also derived a lower and upper bounds for the actual minimum number of bits needed to encode the states in the protected FSM. The lower bound is found to be very tight and actually is equal to the minimum

number of bits used in state encoding when the number of protected states is a power of 2.

The analysis of the problem suggests that the hamming distance between the protected states codes should be at least 3 to be able to add redundancy to them without code overlapping between their redundant states.

We found that there is a strong relation between the state probability based protection using algorithm 1 and the failure rate or fault tolerance. In other words, the failure rate of sequential circuits which involves protecting states with high probability of occurrence is less than the ones involving protecting random or lower state probability.

According to our knowledge, none has presented the introduction of redundant states for states with high probability of occurrence at the FSM level. This idea introduces a new class of methods that tackle the soft error effect in early stages, namely the design stage.

CHAPTER 4

ENHANCING RELIABILITY TO AREA OPTIMIZED SEQUENTIAL CIRCUITS

It is well known that state assignment has a significant impact on several design parameters including area, power and testability. In this chapter, we explore the impact of state assignment on the fault tolerance of sequential circuits. Then, we propose an algorithm, Algorithm 2, which enhances reliability to area optimized sequential circuit.

In the following sections, we start by inspecting the impact of state assignment on sequential circuit reliability. As will be demonstrated, changing state assignment has low impact on increasing fault tolerance. Hence, we target optimizing one criteria, area, in sequential circuits. After that, we will introduce our proposed algorithm which enhances reliability and optimizes area for sequen-

tial circuits and discuss it in details. Eventually, the framework that should be followed by adopting our method is presented.

4.1 Impact of State Assignment on Sequential Circuit Reliability

The computational complexity for finding the best state assignment that gives the best reliability for a sequential circuit is an NP-hard problem [58, 74–77]. There are huge number of possibilities to choose from in order to assign codes to states. For example, assume that in the benchmark circuit dk14 we want to protect 2 states out of 7 states. According to Algorithm 1, the minimum number of bits needed to encode them is 4 bits. we choose from the possible 16 bit strings 7 codes to encode 7 states. In order to find what is the best encoding in this case, the number of choices we have is given by: $\frac{p!}{(p-s)!}$, where p is the possible number of bit strings available to encode the states; and s is the number of states. Thus, in our example, the resulting number of possible ways to encode the states is $\frac{16!}{9!}$ which is more than 57 million ways.

If we take into account the distance-3 relation between the protected states, the number of ways can be reduced. Form the 4 bits required to encode the states, we choose 3 bits to be complemented in the other state code. This way we ensure the distance-3 relation. Hence, there are $\binom{4}{3}\ 8 = 4 * 8 = 32$ ways to assign codes to the protected states. For each way of them, there are 8 associated redundant

states. So, the remaining 5 normal states are permuted out of $14 - 8 = 6$ possible codes. Therefore, the total number of ways is reduced to $32 * \frac{6!}{1!} = 32*720 = 23040$ ways. Although 23040 is a small number of ways compared to 57 million, it is still a large number of experimental tests to point out the best encoding among them.

Actually Algorithm 1 reported in Section 3.3 assumes that the first protected state code is going to be *all zeros* and the normal states are found by *first found first chosen*. Hence, there is only one way to assign states using Algorithm 1. Using the codes produced from Algorithm 1 compared with 4 other different codes, it is found that the impact of changing the state assignment to the failure rate (i.e. reliability) is minimal. For example, Table 4.1 shows 5 different codes used by protecting 4 states out of 7 in dk14 FSM. Algorithm 1 produces codes with the minimum number of bits used in the encoding, 5 bits in our case. The other 4 codes are chosen randomly and happen to be codes of 6 bits length. Notice that all these codes satisfy the hamming distance constraint between the top 4 states that are chosen for protection (i.e. the hamming distance between the protected states is at least 3). Hence, all single faults are tolerated if the FSM is in one of the protected states.

Regardless of their code length, the failure rate for all the codes is very close to each other as shown in Figure 4.1. They are compared to the original circuit without adding protection. The failure rate is greatly reduced compared to the original circuit. However, all the 5 codes have almost the same failure rate. This is explained by the behavior of the FSM of dk14 after adding the protection which

Table 4.1: Different state assignment for dk14.

states		dk14		codes after protection				
		prob.	accum.	code 1	code 2	code 3	code 4	algorithm 1
Protected	S3	2.43E-01	2.43E-01	101010	101101	001001	011101	00000
	S1	1.88E-01	4.31E-01	000000	000111	000111	000111	00111
	S2	1.88E-01	6.19E-01	110001	110110	010000	100001	11001
	S5	1.87E-01	8.06E-01	011100	011011	111101	001010	11110
Normal	S4	1.25E-01	9.31E-01	000011	000100	001110	000100	01010
	S6	5.38E-02	9.85E-01	000101	000010	000010	010110	01011
	S7	1.57E-02	1.00E+00	000110	000001	000100	000000	01100
				6 bits	6 bits	6 bits	6 bits	5 bits

is not related to the state assignment. Similar behavior is observed for sequential benchmark circuit *bbara* which will be discussed in Section 6.4.

4.2 Area Optimization in Sequential Circuits

Since state assignment has low impact on reliability, there is no point of trying to find the best state encoding or assignment that gives the best reliability. Instead, it is better that we target optimizing area in sequential circuits and then enhance its reliability. To optimize area for sequential circuit, state minimization and state encoding play an important role.

After *state minimization* is performed, a set containing minimum number of states is produced representing the finite state machine (FSM) on hand. Encoding the resulting states affects both the number of storage elements used as well as the combinational logic. Encoding for finite state machines has traditionally been targeted for reducing the complexity of its combinational part. A good survey of FSM encoding for area can be found in [78]. The number of possible combinations to encode FSM is exhaustively large. It is given by: $\frac{2^{nb}!}{(2^{nb}-s)!}$, where nb is the number of bits used to encode the states and s is the number of states themselves.

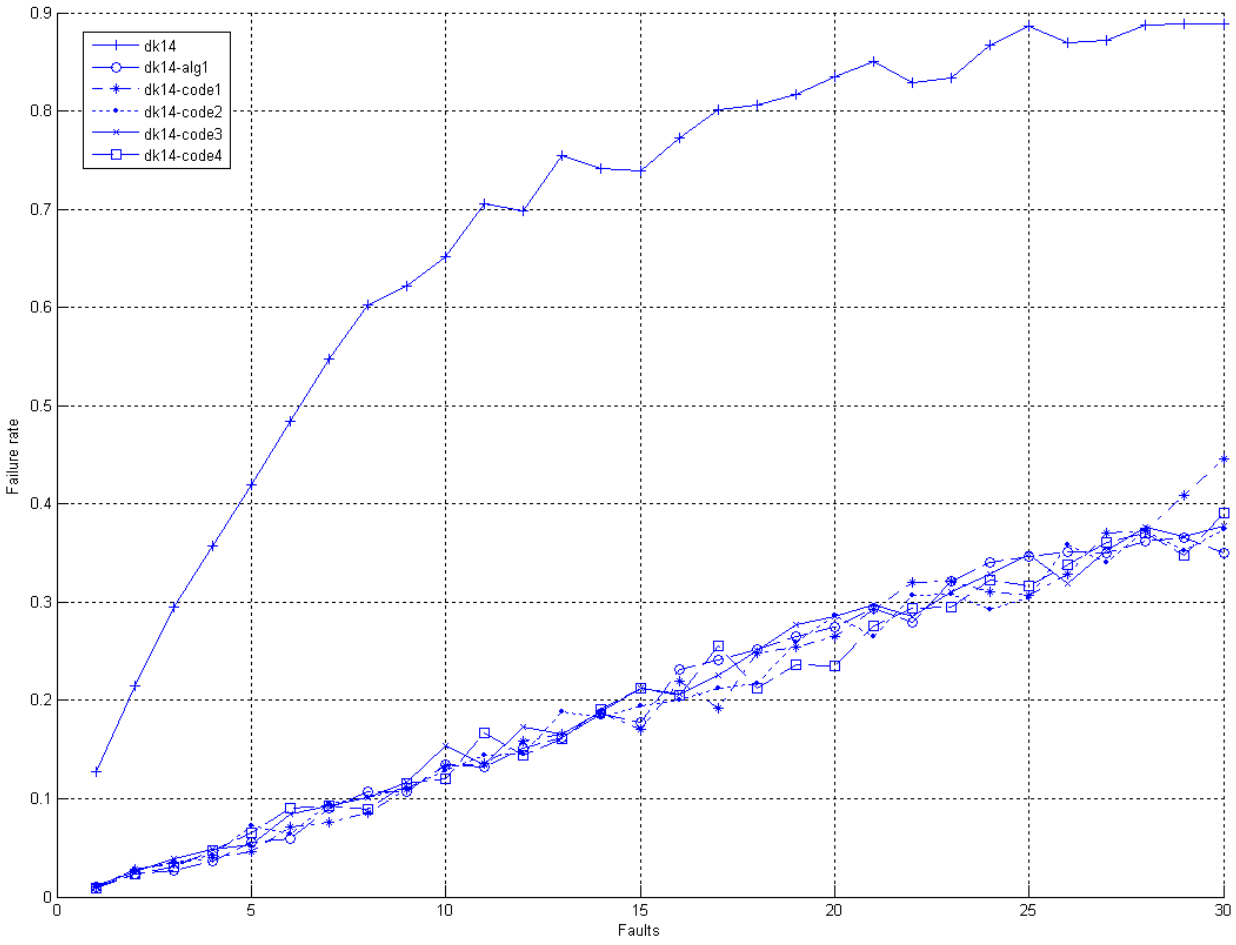


Figure 4.1: Impact of state assignment on sequential circuit reliability.

Thus, exhaustive evaluation is not feasible and heuristics are generally employed to tackle the problem of FSM encoding.

One of the famous methods for state encoding to optimize area is symbolic minimization [79] which will be discussed in the next sub-section. Taking advantage of symbolic minimization, nova [80] reduces the area of sequential circuits and is adopted in our work.

4.2.1 Two Level Symbolic Minimization and Multiple Level Models

Logic minimization aims to optimize the area of sequential circuit by optimizing the combinational logic of an FSM. This in turn depends on the degree of freedom provided by an efficient FSM encoding. A good encoding can help the logic minimizer to achieve a better realization in terms of logic cost. Logic minimizers employ different heuristics for two-level and multilevel circuits as their cost measures differ.

A two level implementation realizes a logic function as a sum of product terms. The circuit complexity of such a representation is related to the number of inputs, outputs, number of product terms and number of variables utilized in a product term, i.e. the number of literals.

The simplest way to encode an FSM is by assigning 1-hot state codes. In 1-hot encoding for a state, the corresponding code bit for a state is set to 1 and all others to 0. Thus, 1-hot encoding is a case of non-minimal state assignment

Table 4.2: Implicant merging.

PS	I	NS	z
S_1 000	i	S	o
S_2 001	i	S	o
S_3 010	i	S	o
S_1,S_2,S_3 0--	i	S	o

such that the number of variables required is equal to the number of states. It is further noticed in [79,81] that such an encoding is poor to minimizing the size in sum of products representation.

The complexity of a two level realization can be reduced by using mechanisms such as *implicant merging*, *code covering* and *disjunctive coding* [78]. The idea behind Implicant Merging is to assign adjacent codes to states that produce either the same next-state or output or both at similar input conditions. This yields bigger cubes while doing Karnaugh minimization, and results in a simpler final expression. Implicant merging, as shown in Table 4.2, requires adjacency constraints to be met by the state assignment algorithm. In Table 4.2, S_1, S_2, S_3 are merged into one state with code 0– because they have the same input, next state and output.

Code Covering involves a code word of a state covering a code word of some other state(s), i.e. all the bit positions for which the second code word is 1, correspond to 1 in the first code word. An example utilizing code covering is

Table 4.3: Code covering.

PS	I	NS	z
S	001	S_1 110	o
S	000	S_2 100	o
S	01-	S_2 100	o
S	001	S_1 110	o
S	0--	S_2 100	o

illustrated in Table 4.3. Assume that S1 is encoded with 110 and S2 with 100. In this case, the input condition (s, 001) can be treated as don't care condition for the next state S2, reducing the cover cardinality from three to two. Covering constraints produce covering codewords.

Reducing cover cardinality using Disjunctive Coding is illustrated in Table 4.4. Disjunctive constraints require that the disjunction of state codes is equal to some other state code. In the example shown, the states are encoded such that the code for S2 is the disjunction of the state codes for S1 and S3. As such, the second implicant with the input field 101 gets contained in other input conditions and thus is completely saved.

The problem with many approaches to two-level assignment is that no exact predictions are possible, as to how the satisfaction of coding conditions affects the complexity of the resulting combinational logic, since the different coding conditions interact with each other in a complex way. The application of coding constraints and finding out their effect would be excessively costly as it would

Table 4.4: Disjunctive coding.

PS	I	NS	z
S	001	S_1 100	o
S	101	S_2 110	o
S	111	S_3 010	o
S	-01	S_1 100	o
S	1-1	S_3 010	o

require a huge number of logic minimizations.

To mitigate this problem, [79] proposed an elegant solution of *symbolic minimization*. Symbolic minimization has been proposed to take into account the effect of the encoding on the next state part. It is a technique that yields a minimal encoding-independent sum-of-products representation of a symbolic function. It builds up a directed acyclic graph (DAG), where the nodes are the next states and an edge (u, v) corresponds to the covering constraint (called output constraint) that the code of u covers bit-wise the code of v. The translation of the cover obtained by symbolic minimization into a *compatible* Boolean representation defines simultaneously a face *hypercube embedding problem* and an *output covering problem* (called ordered face hypercube embedding). Moreover, output minimization techniques may be seen as setting disjunctive constraints on the codes of the symbolic states (the code of some states is the logical disjunction of the codes of two or more other states). Finding a compatible Boolean representation entails solving a difficult encoding problem based on input, output, and disjunctive constraints.

Therefore, by using symbolic minimization techniques, it is possible to opti-

mize the function independently of the encoding and determine the codes at a later time. This requires performing the minimization at the symbolic level, before the encoding. As a summery, symbolic minimization is actually a mixed encoding for both input and output. It takes advantage of *implicant merging*, *code covering* and *disjunctive coding* to minimize the circuit on hand while holding the resulting constraint from those relations which is input constrains, output constrains, covering constraints and disjunctive constraints.

In contrast to two level circuits, multiple level circuits provide much more degree of freedom in optimizing combinational network and satisfying coding constraints. This is because of the flexibility provided due to operations such as common subexpression extraction and factorization. Unfortunately, it comes with an increase in the difficulty of modeling and optimizing the multilevel network themselves. The complexity measure for multilevel circuits is the encoding length and the number of literals in the optimized logic network. Since encoding length is mostly taken constant, literal saving by extracting common subexpressions has been the focus of most of the work done for multilevel FSM optimization. This involves finding the state pairs which when encoded carefully can result in extracting common subexpressions. In contrast to two level circuits, state pairs in multilevel implementations do not necessarily have to be given adjacent codes. If two states have n state bits in common, the combination of the two states result in a common subexpression with n literals. To identify the states to assign close codes, two heuristics proposed by [82, 83] standout. The first called fanout

oriented, tries to assign closer codes to the states that have same next state transition. The rationale is to maximize the size of common cube by assigning closer codes (lesser hamming distance) to such states. The second approach is referred to as fan-in oriented in which state pairs with incoming transitions from the same states are given high weights for closer code assignment. Here the motivation is to maximize the frequency of common cubes in the encoded next state function. Rules for detecting potential common cubes for more precise evaluation of literal savings have been proposed in [84].

4.2.2 State Assignment Using Nova

In [80] heuristics for optimal state assignment of FSMs based on the solution of *face hypercube embedding* and *ordered face hypercube embedding* have been proposed. Based on symbolic minimization, the proposed theoretical framework offers substantial advantages over previous approaches to develop effective algorithms. The algorithms are part of *nova*, a program for optimal encoding of control logic, available as a tool of the Berkeley logic synthesis system [85, 86]. Some of the heuristics that are part of nova perform the following:

- Solve *face hypercube embedding* problem.
- Propose an exact algorithm that finds an encoding satisfying all input constraints and minimizing the encoding length.
- Propose heuristic encoding algorithms that maximize input constraint satisfaction.

- Present heuristic encoding algorithm that maximizes simultaneous input and output constraint satisfaction, according to an appropriately defined metric.

For detailed information and pseudo codes of nova's heuristics and algorithms mentioned above, the reader can access nova detailed paper in [80].

Although nova targets two-level logical implementations, it is found that the final literal counts in a factored form of the logic when encoded by nova are 30% less than the literal counts obtained by the best of a number of random state assignments. Comparisons with Mustang [83] in the two-level and multilevel case are also reported. Even though nova was not designed as a multilevel state assignment program, its performance compares successfully with Mustang. Hence, nova is adopted in our work.

4.3 Algorithm 2: Enhancing Reliability and Optimizing Area for Sequential Circuits

Given an area optimized sequential circuit with a certain obtained state assignment using nova, Algorithm 2 builds fault tolerance layer on top of that optimization. Algorithm 2 (see Table 4.5 and Table 4.6) protects high probable states by introducing associated equivalent redundant states to them the same way Algorithm 1 did. Instead of starting from empty set, Algorithm 2 takes the state encoding resulting from a certain enhancing heuristic, nova in our case, and append bits to the left of the codes until a distance 3 relation between the protected

states is obtained.

Algorithm 2 is divided into the following parts: *getting the protected states codes*, *checking if the normal states are covered by the remaining codes based on the current number of bits used to encode them*, *getting the redundant states codes* and finally *getting the normal states codes*.

The inputs are the state codes achieved by optimizing the sequential circuit for a certain constraint, area for example. They include the protected state codes and normal states codes. In the initial stage, the current number of bits used to encode the states is set to the number of bits used by the given codes. Also the redundant states set is initialized to empty set.

Starting with the protected states codes given as an input, we append them from the left bit by bit. By appending each one of them by 1 bit, we form a bit vector constructed from all the bits that are added to them. Then, we choose the best bit vector that gives the maximum number of distance-3 codes in the set of the protected state codes. While the hamming distance between all the protected states is less than 3, we append or pad the current protected state codes with 1 bit and search for the best distance between codes that are guaranteed by the best bit vector. For example, shown in Figure 4.2 are 2 bit vectors which are added to r protected states. They are $v_t = B_{0t}, B_{1t}, ..., B_{rt}$ and $v_{t+1} = B_{0t+1}, B_{1t+1}, ..., B_{rt+1}$. Each bit B can be either 0 or 1 depending on the the best distance codes.

The heuristic by which we get the protected states codes is further explained by the function *get_best_distance_codes()* shown in Table 4.6. This function is

called with the current protected state codes and the current number of bits used to encode them as arguments. The bit vector can have any binary bit string from 0 to $2^{Numberofcodes} - 1$. We inspect the current bit vector i by calculating the hamming distance between the protected codes after padding them with i the same way as shown in Table 4.2. For each i, we calculate how many codes with distance-3 or above as well as codes with distance-2 as a cost counters. The bit vector with the most distance-3 codes is selected. In addition, if 2 or more bit vectors have the same number of distance-3 codes, the one with more distance-2 codes is selected. We keep calling *get_best_distance_codes()* as long as the distance between all the protected state codes is less than 3.

After we get the protected state codes, the remaining codes and the redundant state codes are obtained in the same manner as explained in Algorithm 1 in Section 3.3. The remaining codes is $2^{\alpha+\beta} - n(1 + \alpha + \beta)$ where $\alpha + \beta$ is the current number of bits used in the encoding of the protected states as well as their redundant states. The additional number of bits needed to cover all the normal states, γ, is also obtained in the same manner as it is derived in Algorithm 1. However, getting the normal states codes slightly differs from how they are obtained in Algorithm 1. In Algorithm 1, codes that are not used for protected or redundant states are simply picked for normal states. On the other hand, normal state codes are already given as inputs in Algorithm 2. However, since protected states codes are computed by padding bits to the left until 3 hamming distance is obtained, normal state codes must be padded with the same number of bits

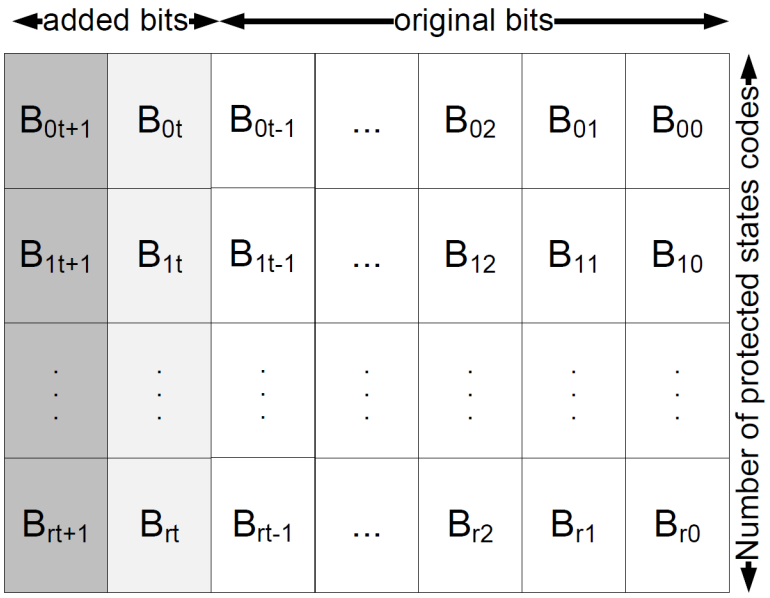


Figure 4.2: Two bit vectors added to protected states codes.

as the protected states. Therefore, we check the bit strings that result from the *concatenation* between each of the original normal state codes and i where i can be any binary string from 0 to 2^a where a is the the number of the left padded bits that makes the normal state codes length equal to the number of bits used for protected and redundant state codes. If the resulting bit string is not used for protected states or redundant states, then it is assigned to one of the normal states.

The outputs of Algorithm 2 are the protected states codes, the redundant states codes, normal states codes and the minimum number of bits used to encode all of them as shown in Table 4.5.

Table 4.5: Algorithm 2.

Inputs	• Get the *protected states codes* and the *normal states codes*, which optimize the sequential circuit for area, as inputs.
Initialize	• Let the *number of bits used* = length of one of the *protected state codes*.
Protected states	• While the hamming distance between the *protected state codes* is less than 3: – Increment the current *number of bits used* by 1. – Find the best bit vector padding to the *protected state codes* using get_best_distance_codes().
Find γ	• Find the extra number of bits needed to encode all the *normal states*, γ. • Increment the current *number of bits used* by γ.
Redundant states	• For each *protected state*, assign all hamming-distance-1 codes as its *redundant state codes*.
Normal states	• Check the concatenation of each of the original normal state codes and i where i can be any binary string from 0 to $2^a - 1$ where a is the number of the left padded bits that makes the *normal state codes* length equal to the current *number of bits used*. If it is not used for *protected states* or *redundant states*, assign it to one of the *normal states* until all *normal states* are covered.
Outputs	• Collect *protected state codes*, *redundant state codes*, *normal state codes* and *number of bits used* as outputs.

Table 4.6: Algorithm 2 - get_best_distance_codes.

Get best distance codes	Get_best_distance_codes() returns the set of *protected state codes* which has the best hamming distance between them after padding them with the best bit vector, as follows: • Let the *best bit vector* = all zeros. • Run through all bit vectors indexed by i. – Check hamming distances between *protected state codes* after temporally padding them with the bit vector i. – Evaluate the current bit vector i based on the hamming distance between the *protected state codes*. The bit vector i with the most hamming distance 3 pairs of protected state codes is selected. Ties are broken by counting the hamming distance 2 codes pairs. – If i is better than the current *best bit vector*, bit vector i replaces it. • Pad the *protected state codes* with the *best bit vector* from the left. • Return the *protected state codes*.

4.3.1 Complexity of Algorithm 2

Finding protected states codes are the key point to compute the complexity of Algorithm 2. To find them, we run through 4 nested loops. In the outer loop as shown in Table 4.5, we keep padding bits to the state codes in each iteration to the point where the overall distance between all the protected states codes are 3 or above. This operation takes $N - M$ iterations where N is the *resulting* number of bits used to encode the protected state codes and M is the *original* number of bits used to encode the given protected state as inputs. In order to find the best bit vector in each iteration, we run through all the possible combinations formed by the bit vectors from 0 to $2^K - 1$ (see Table 4.6), where K is the number of the given protected state codes. For each bit vector, the distance between the protected state codes are checked. The checking operation takes $O(K^2)$ iterations. Therefore, the overall complexity of Algorithm 2 is $O((N - M)2^K K^2)$.

4.3.2 Expected Area Overhead of Algorithm 2

The resulting area overhead is kept minimal because we start from an optimized state codes using nova. The reason of this is analyzed by the corresponding equations of the D type flip flops that are used to build the sequential circuit. The equations of the flip flops are represented in a sum of products form which contains a set of cubes. The area overhead in the protected sequential circuit is related to the original set of cubes that are resulting from optimizing its area. Since we started from an optimized set of cubes that gives a minimized area, it

is expected to give a local minimum in the protected version of the sequential circuit. Actually, the protected D type flip flop equations of the fault tolerant circuit are generated by the following rules:

1. A cube C in the original equation of an FSM will remain as is in the equation of the protected FSM unless:

   (a) The cube C overlaps with one of the redundant states of other protected states not covered by the cube.

   (b) The cube C overlaps with a code X in the original FSM that was assigned a don't care and in the protected FSM is assigned to a redundant state that assigns its value to 0.

   The cube C will become C # (overlapped codes) where # is the sharp operator. This will result in additional literal(s) and/or splitting the cube C into 2 or more cubes.

2. For each cube C, extra cubes will be added to cover the codes for redundant states that are not covered by C.

3. Cubes in the newly added state variables due to protection are mostly cubes from other original state variables or reduced version of them.

These rules are explained by two examples. The FSM of the first example is shown in Figure 4.3. It contains 2 states originally coded as shown in Table 4.7. After minimization, the equations for the state variable $v1$ and output are (See

K-map in Table 4.8):

$v1 = a\bar{v}1 + \bar{a}v1;$

$z1 = a\bar{v}1 + \bar{a}v1;$

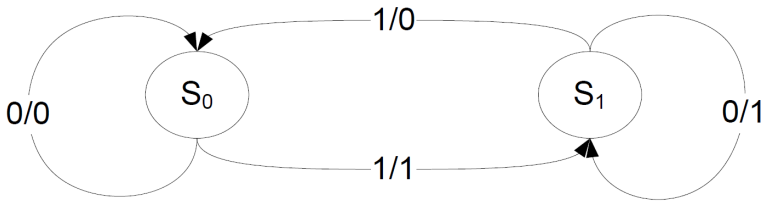
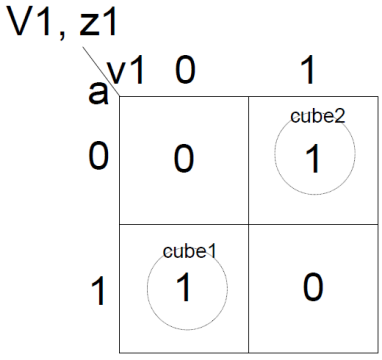


Figure 4.3: FSM example with 2 states.

Table 4.7: State codes of the original FSM shown in Figure 4.3.

State	Code v1
S0	0
S1	1

Table 4.8: FF and output function cubes shown in K-map of the original FSM shown in Figure 4.3.



Assuming that we protect both S0 and S1 by applying Algorithm 2 detailed in Section 4.3, we obtain the protected states codes shown in Table 4.9. After minimizing, the equations for the state variables $v1$, $v2$, $v3$ and output are (See K-map in Table 4.10):

$v1 = a\bar{v}1\bar{v}2 + a\bar{v}1\bar{v}3 + \bar{a}v1v2 + \bar{a}v1v3 + a\bar{v}2\bar{v}3 + \bar{a}v2v3;$

$v2 = a\bar{v}1\bar{v}2 + a\bar{v}1\bar{v}3 + \bar{a}v1v2 + \bar{a}v1v3 + a\bar{v}2\bar{v}3 + \bar{a}v2v3;$

$v3 = a\bar{v}1\bar{v}2 + a\bar{v}1\bar{v}3 + \bar{a}v1v2 + \bar{a}v1v3 + a\bar{v}2\bar{v}3 + \bar{a}v2v3;$

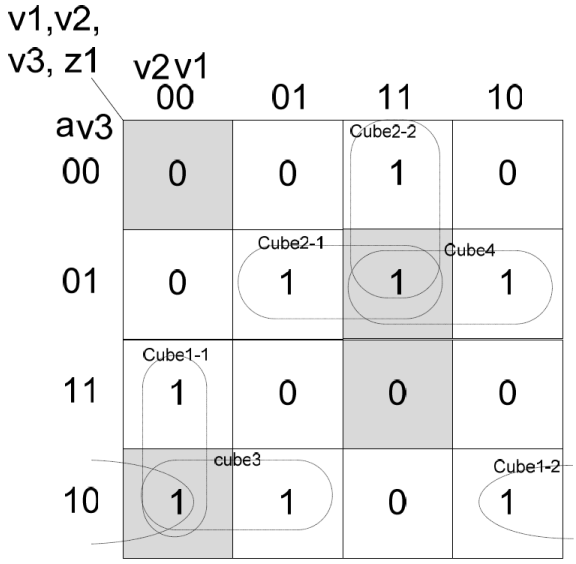
$z = a\bar{v}1\bar{v}2 + a\bar{v}1\bar{v}3 + \bar{a}v1v2 + \bar{a}v1v3 + a\bar{v}2\bar{v}3 + \bar{a}v2v3;$

Table 4.9: State codes of the protected FSM shown in Figure 4.3.

	State	code		
		v3	v2	v1
protected	S0	0	0	0
protected	S1	1	1	1

By applying the rules, cube 1 ($a\bar{v}1$) in the original equation of $v1$ is sharped (using # operation) into 2 cubes, cube 1-1 ($a\bar{v}1\bar{v}2$) and cube 1-2 ($a\bar{v}1\bar{v}3$); likewise, cube 2 ($\bar{a}v1$) is sharped into, cube 2-1 ($\bar{a}v1v3$) and cube 2-2 ($\bar{a}v1v2$). The reason for that is explained in condition 1(a). Cube 1 ($a\bar{v}1$) is overlapping with the redundant state with the code (110), so it is sharped. In the same manner, cube 2 ($\bar{a}v1$) is overlapping with the redundant state with the code (001). As stated in condition 2, cube 3 ($a\bar{v}2\bar{v}3$) and cube 4 ($\bar{a}v2v3$) in the protected equation are

Table 4.10: FF and output function cubes shown in K-map of the protected FSM shown in Figure 4.3.



added since they are not covered by cube 1 ($a\bar{v}1$) and cube 2 ($\bar{a}v1$) in the original equation, respectively. Cube 3 ($a\bar{v}2\bar{v}3$) is added to cover the condition for the redundant states (001) and cube 4 ($\bar{a}v2v3$) is added to cover the condition for the redundant state (110). The same rules apply for the output function z.

As a second example, Figure 4.4 shows a FSM of a sequential circuit with 3 states. The original state codes are shown in Table 4.11. After minimization, the equations for the state variables $v1$, $v2$ and output are (See K-map in Table 4.12):

$v1 = a\bar{v}1 + \bar{a}v1 + v2;$

$v2 = a\bar{v}1 + \bar{a}v2;$

$z1 = a;$

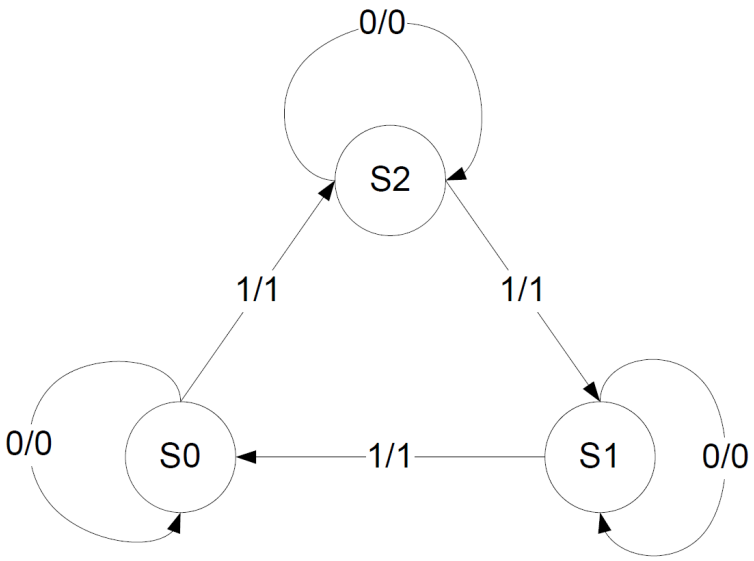
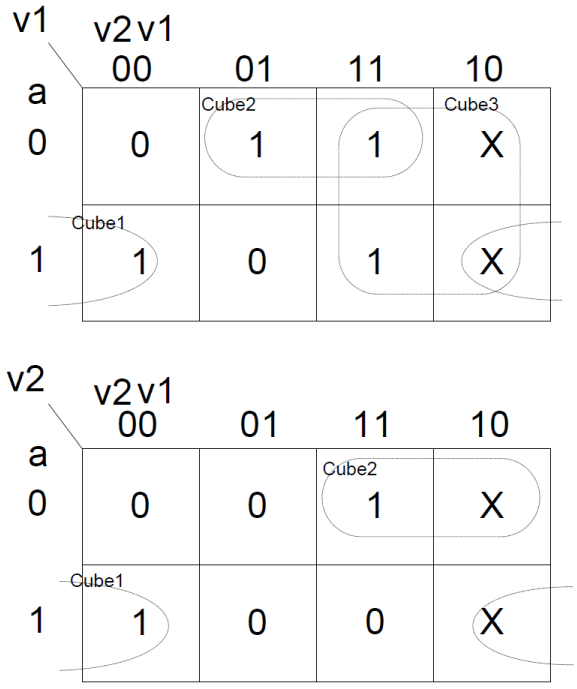


Figure 4.4: FSM example with 3 states.

Table 4.11: State codes of the original FSM shown in Figure 4.4.

State	code	
	v2	v1
S0	0	0
S1	1	1
S2	0	1

Table 4.12: FF and output function cubes shown in K-map of the original FSM shown in Figure 4.4.



Applying Algorithm 2 to protect S0, we get the state encoding shown in Table 4.13. After adding protection to S0, the equations for the state variables $v1$, $v2$, $v3$ and output are (See K-map in Tables 4.14, 4.15 and 4.16):

$v1 = a\bar{v1} + \bar{a}v1v3 + v1v2 + a\bar{v3}$;

$v2 = a\bar{v1} + \bar{a}v1v2 + a\bar{v2}\bar{v3}$;

$v3 = av1v2 + \bar{a}v1v3$;

$z1 = a$;

For state variable $v1$, cube 1 ($a\bar{v1}$) is kept the same as it does not overlap with other protected states or redundant states codes. Cube 2 ($\bar{a}v1$) of the original FSM is sharped into cube 2 ($\bar{a}v1v3$) in the protected FSM which has 1 extra literal as

stated in 1(a) because cube 2 ($\bar{a}v1$) overlaps with the redundant state with the code (001). However, cube 3 ($v2$) in the original circuit is sharped by adding 1 extra literal to cube 3 ($v1v2$) in the protected equation because it overlapped with the code (010) which is assigned to don't care at the original code (10) in the original FSM and to 0 in the protected FSM. This is stated in condition 1(b). Cube 4 ($av\bar{3}$) is added to take care of the redundant state (001) as stated in condition 2. To complete the picture for state variable $v2$, cube 1 ($av\bar{1}$) is kept the same as it does not overlap with other protected states or redundant states codes. Cube 2 ($\bar{a}v2$) in the original FSM is sharped into cube 2 ($\bar{a}v1v2$) in the protected FSM because cube 2 ($\bar{a}v2$) overlaps with the redundant state with the code (010). Finally, redundant state (001) is covered by cube 3 ($av\bar{2}v\bar{3}$).

Table 4.13: State codes of the protected FSM shown in Figure 4.4.

	State	code		
		v3	v2	v1
protected	S0	0	0	0
	S1	0	1	1
	S2	1	0	1

As mentioned in condition 3, in both example 1 and 2, the cubes in the equations of the extra state variables, which are added for protection, are also cubes in the other state variables equations or a reduced version of them. In example 1, equations of $v2$ and $v3$ contain the same cubes contained in the equation of $v1$.

Table 4.14: State variable v1 function cubes shown in K-map of the original FSM shown in Figure 4.4.

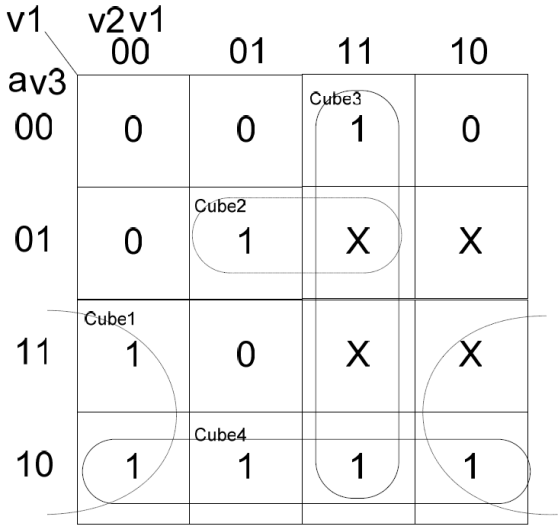
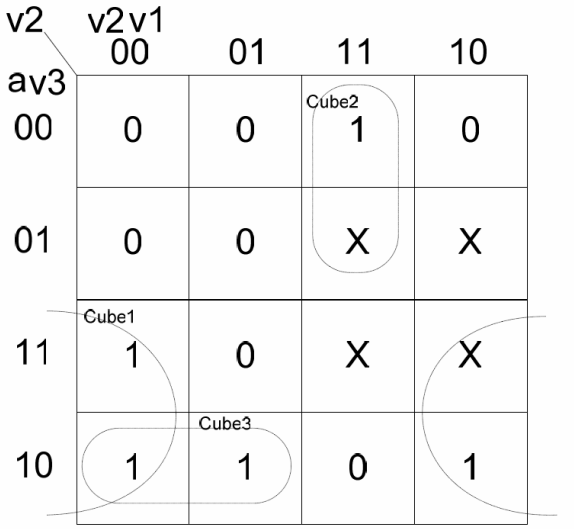


Table 4.15: State variable v2 function cubes shown in K-map of the original FSM shown in Figure 4.4.



Also in example 2, $v3$ has cube 2 ($\bar{a}v1v3$) in the protected FSM which is cube 2 ($\bar{a}v1v3$) in $v1$; and it has cube 1 ($av1v2$) which is a reduced version of cube 3 ($v1v2$) in $v1$.

As explained earlier, Algorithm 2 suggests adding extra bits to the original codes which ensure hamming distance of 3 between the protected states. In example 1, this addition converts the code of the original state $S0$ from (0) to (000) and $S1$ from (1) to (111) as shown in Tables 4.7 and 4.9. This conversion leads to add 2 more state variables, $v2$ and $v3$, with the same truth values as the original state variable $v1$ (i.e. the columns of $v3$ and $v2$ in Table 4.9 are the same as column $v1$). Hence, the D-type FF equations of $v2$ and $v3$ is directly affected in state table level and takes the same cubes in $v1$ equation. In example 2, Tables 4.11 and 4.13 show that the additional state variable $v3$ is covered by state variable $v1$ (i.e. column $v3$ in Table 4.13 is covered by column $v1$) which explains the fact that $v3$ has cube(s) from $v1$ or reduced version of them.

Although the newly added state variables can have cubes from other original state variables or reduced version of them as seen in example 1 and 2 as stated in condition 1, it is not the general case for all sequential circuits. Because an additional state variable can have truth values (i.e. (0)s and (1)s) that are not covered by other state variables. Fortunately, the similarity and coverage relations are very likely to happen between additional and original state variables because of the hamming distance relation between the protected states which will directly affect the D-type FF equations of the new state variables.

4.4 Framework of Enhancing Fault Tolerance Using Algorithm 2

Algorithm 2 protects a sequential circuit that is already optimized for a certain criteria. Taking area as an example, the following framework should be applied in the design procedure, as shown in Figure 4.5:

1. Given a sequential circuit represented by a state table or finite state machine (FSM).

2. Assign states codes for the FSM taking into account optimizing the criteria on hand (i.e. area).

3. Calculate the steady state probability of all the states in the FSM.

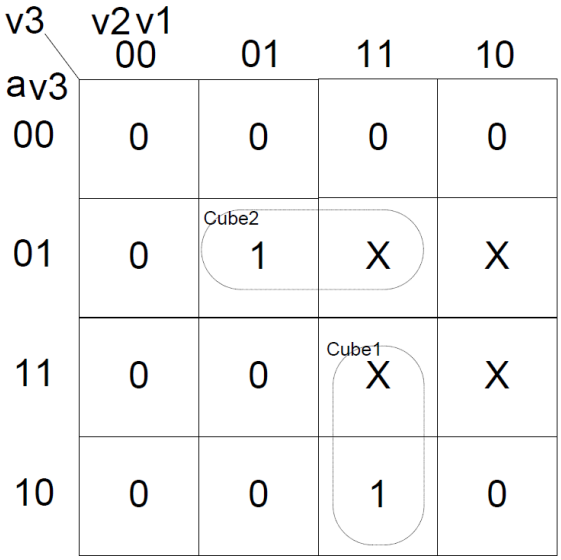
4. Protect the chosen high probable states using Algorithm 2 by introducing redundant states to them as shown in Section 4.3.

5. Synthesis/implement the sequential circuit.

4.5 Conclusion

In this chapter, state assignment is explored and found to have a minimal impact on soft error tolerance of sequential circuits. Hence, Algorithm 2 is proposed by starting from a state assignment or state encoding that optimizes a certain

Table 4.16: State variable v3 function cubes shown in K-map of the original FSM shown in Figure 4.4.



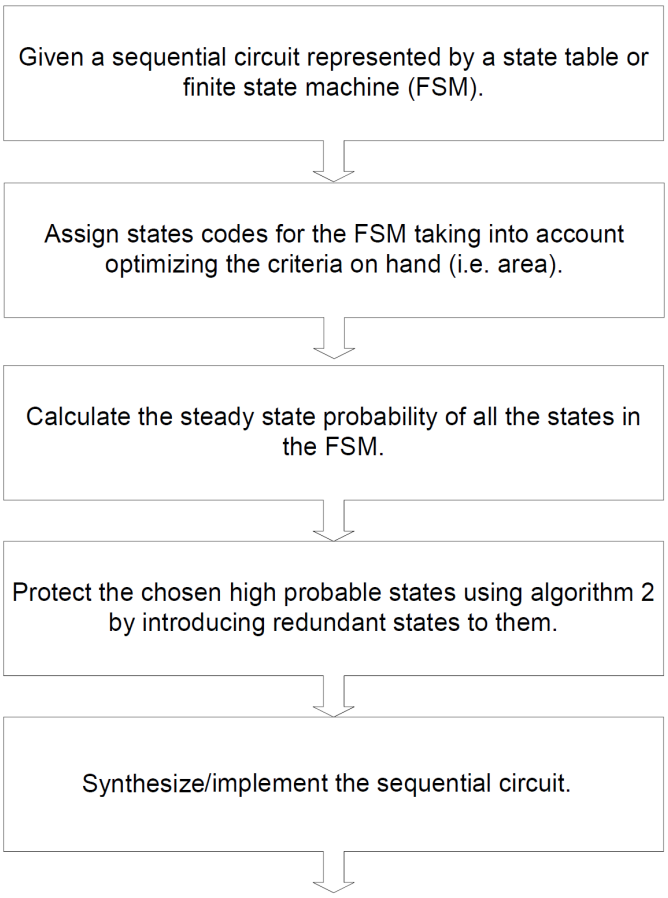


Figure 4.5: Framework of Algorithm 2.

criteria such as area or power. Fault tolerance can be added on top of the initial state assignment and the optimization is kept.

An optimized sequential circuit for area using nova can be enhanced by adding a layer of protection using state redundancy by adding extra bits to the left of chosen protected states encoding that have high probability of occurrence. As the original encoding of the protected states optimizes area on one hand, the extra added bits, which guarantee a hamming distance of 3 between the protected states, result in fault tolerance property on the other hand.

Since we started from a minimum point of area overhead which results from the original state encoding, the resulting area overhead in the protected circuit remains minimal. This is explained by the fact that cubes from the original FSM are actually either kept the same, result in a reduced form of them (more literals are added), split into two or more cubes, or/and additional cubes are added to cover the redundant states.

CHAPTER 5

SIMULATION ENVIRONMENT AND FRAMEWORK

To show the impact of our proposed algorithms in Chapter 3 and Chapter 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 ISCAS89 benchmark sequential circuits. In the following sections, we will describe how we measure a sequential circuit reliability. Then, a Monte-carlo based simulation is presented which details the framework of our reliability evaluator. 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 explored.

5.1 Measuring Sequential Circuit Reliability

Comparing the reliability of original sequential benchmark circuits with their protected version, 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 sequential 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 injected/occur and decrease as the number of faults increases. In other words, the failure rate grows as the number of faults increases.

Sequential circuits reliability is also directly affected by the logical, electrical and time masking which depends on their actual implementation. Masking properties are reflected by the measurement of failure rate against the number of faults as more masking implies less failure rate. In addition, our technique presents state protection which reduces the failure rate and hence increases reliability. In summary, by plotting fault rate against the number of injected faults in sequential circuits, reliability, masking as well as protection are measured.

For evaluating circuit failure probability and reliability based on unit error probability, a simulation-based reliability model as the one used in [87] is often adopted. However, since we eventually plot the failure rate against the number of injected faults, we modified it to our need as it will be explained in the next section.

5.2 The Simulation Framework of Reliability Evaluator

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

- For every number of faults F from 1 to N, where N is the maximum number of injected faults:
  - Set the number of failed iterations, k, the number of protected iterations, p, the number of logically masked iterations, m, to 0.
  - For each iteration i from 0 to a certain number of iterations S:
    - Generate a random sequence of C clock cycles, where the length of each entry in the random sequence is the same as the number of input in the circuit.
    - Simulate the circuit to get the fault free original output by applying the generated random sequence and store the output sequence in O, where the length of each output in O is the same as the number of outputs in the circuit.

* Inject F number of faults in the circuit randomly in the cycle number $\lceil \frac{C}{2} \rceil$. This way temporal faults are emulated.
* Simulate the circuit to get the faulty output sequence and store it in vector O_F.
* If the output sequence O differs from O_F in any cycle, then the current iteration i which involves injecting F faults failed and k is incremented. Otherwise, it is successful.
* A successful iteration is either protected by redundant states, hence p is incremented; or masked logically within the circuit, hence m is incremented.

- Repeat for next i.
- Calculate the failure rate when F faults are injected, denoted by R_F, by $R_F = \frac{k}{S}$.
- Calculate the protection rate when F faults are injected, denoted by R_P, by $R_P = \frac{p}{S}$.
- Calculate the masking rate when F faults are injected, denoted by R_M, by $R_M = \frac{m}{S}$.
- We can plot R_F, R_P and R_M versus F as needed.

- Repeat for next F.

5.3 Assumptions

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

- Since we are experimenting the effect of temporal soft faults, the number of faults that are injected to the circuit should be few; actually they are measured by FIT (failure in each 10^9 hours of operation). However, they are injected up to $N = 30$ faults.
- The number of iterations, S, is 1000.
- The number of cycles, C, is 33. Hence the faults are injected in cycle number 17.
- The faults are injected randomly based on perl rand() method.
- Faults are injected randomly in the gate level of the circuit.
- Since the proposed techniques affect the protection of faults occurring in FFs or thier logic, the faults are injected in the combinational logic part that represent the D-type flip flop equations or the flip flop themselves. In other words, the combinational logic part corresponding to the output equation is excluded.
- Only logical masking is considered. The electrical masking as well as timing masking are not considered by our evaluator.

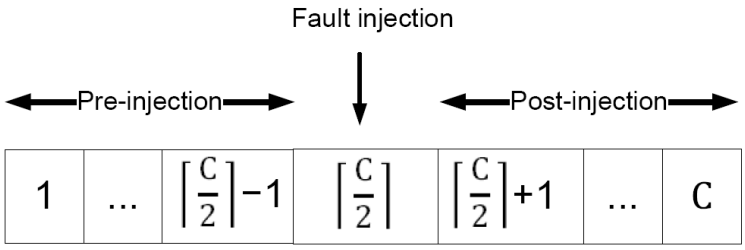


Figure 5.1: Injection traces.

5.4 Fault Model and Fault Injection Mechanism

In our work, we assume a stuck at fault model. When we inject faults at any gate, it can be either stucked-at-1 (i.e. connected to Vdd) or stucked-at-0 (i.e. connected to ground). In order to preserve the temporal property of the faults, each simulation iteration is done for C number of cycles. The stuck-at faults are injected in one cycle (i.e. the middle one).

By injecting faults in the middle cycle, FSM trace of states can be divided into 3 parts as shown in Figure 5.1. *Pre-injection trace* at which the FSM is initialized to a certain state; *injection trace*, which is only 1 cycle at which the faults are injected; and *Post-injecting trace*, at which the circuit is tested whether the fault effect is propagated. According to [29], 10 cycles are enough to test if the faults are affecting the behavior of the FSM or not. In our method, we assume that the *Post-injecting trace* contains 16 cycles.

Fault injection happens in the gate level of the circuit randomly. 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 at the injection cycle. When a stuck-at-0 fault is injected, the line at which the fault is injected is replaced by an AND-gate

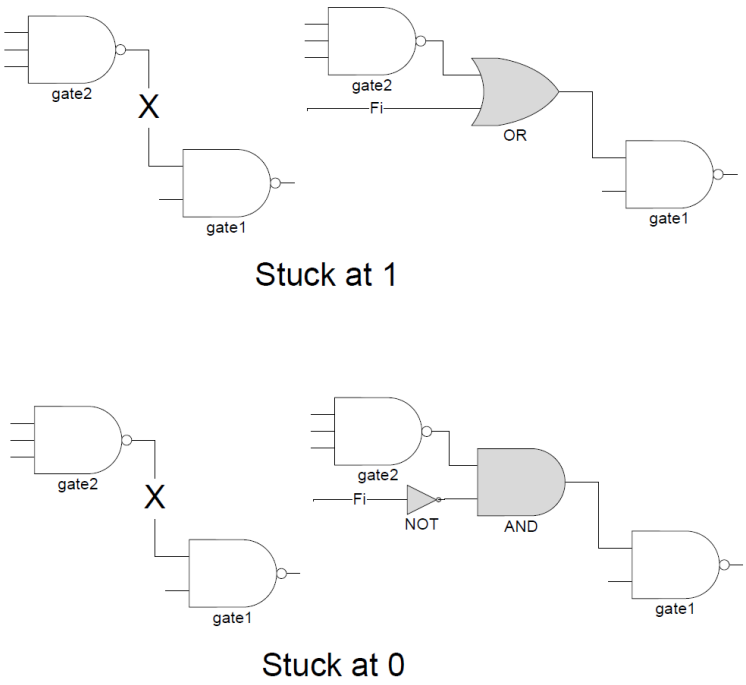


Figure 5.2: Fault injection mechanism.

with an inverted input. The fault indicator F_i is connected to the inverted input and is set to 1 at the injection cycle. This is shown in Figure 5.2.

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 both Algorithm 1 described in Chapter 3 and Algorithm 2 described in Chapter 4.

6.1 Experiments

Several experiments have been performed for different ISCAS89 sequential circuits. The circuits bbara, bbsse, dk14, lion9, shiftreg, train11, cse, keyb, s1, planet, pma, s832, s1494, sand, styr and tbk are the ones chosen for our experiments.

For each circuit, several versions are implemented and experimented as follows:

- Original circuit with sharing logic in the combinational logic blocks implementing the next state equations of memory elements.
- Original circuit without sharing logic in the combinational logic blocks implementing the next state equations of memory elements.
- Protecting a number of states that yields a minimum of 25% state probability coverage without sharing the logic.
- Protecting a number of states that yields a minimum of 50% state probability coverage without sharing the logic.
- Protecting a number of states that yields a minimum of 90% state probability coverage without sharing the logic.
- Protecting all the states.

Tables 6.1, 6.2, 6.3 and 6.4 show the steady state probability analysis for the chosen circuits. For each circuit, 25%, 50% and 90% state probability coverage are marked. In addition, the number of states protected in each case is shown. For example, for the circuit bbara, 90% state probability coverage involves protecting 5 states out of 10. On the other hand, for the circuit cse, 90% state coverage requires protecting only 2 states out of 16.

Two different reliability enhancement methods are done for each circuit as follows:

- Hardening: it covers all the methods that involve protection of faults occurring at the memory elements. In simulation, we assumed the best case

Table 6.1: FSM benchmark steady state probability analysis: bbara, bbsse, dk14 and lion9.

states	bbara		states	bbsse		states	dk14		states	lion9	
	prob.	accum.		prob.	accum.		prob.	accum.		prob.	accum.
S2	2.67E-01	2.67E-01	S12	4.50E-01	4.50E-01	S3	2.43E-01	2.43E-01	S1	1.11E-01	1.11E-01
S5	1.97E-01	4.64E-01	S1	2.25E-01	6.75E-01	S1	1.88E-01	4.31E-01	S2	1.11E-01	2.22E-01
S1	1.55E-01	6.19E-01	S2	2.14E-01	8.89E-01	S2	1.88E-01	6.19E-01	S3	1.11E-01	3.33E-01
S3	1.33E-01	7.52E-01	S13	7.50E-02	9.64E-01	S5	1.87E-01	8.06E-01	S4	1.11E-01	4.44E-01
S4	1.33E-01	8.85E-01	S5	1.45E-02	9.79E-01	S4	1.25E-01	9.31E-01	S5	1.11E-01	5.55E-01
S6	4.92E-02	9.34E-01	S3	1.38E-02	9.92E-01	S6	5.38E-02	9.85E-01	S6	1.11E-01	6.66E-01
S8	3.74E-02	9.72E-01	S6	3.63E-03	9.96E-01	S7	1.57E-02	1.00E+00	S7	1.11E-01	7.77E-01
S7	1.64E-02	9.88E-01	S4	3.46E-03	9.99E-01				S8	1.11E-01	8.88E-01
S9	9.36E-03	9.97E-01	S7	2.59E-04	1.00E+00				S9	1.11E-01	9.99E-01
s10	2.34E-03	1.00E+00	S8	3.24E-05	1.00E+00						
			S9	9.25E-06	1.00E+00						
			s10	1.18E-06	1.00E+00						
			S11	3.14E-07	1.00E+00						
count	10		count	13		count	7		count	9	
25%	1		25%	1		25%	1		25%	2	
50%	3		50%	2		50%	3		50%	5	
90%	5		90%	3		90%	4		90%	8	

Table 6.2: FSM benchmark steady state probability analysis: train11, cse, keyb and s1.

states	train11		states	cse		states	keyb		states	s1	
	prob.	accum.		prob.	accum.		prob.	accum.		prob.	accum.
S1	1.67E-01	1.67E-01	S1	8.65E-01	8.65E-01	S1	7.22E-01	7.22E-01	S18	1.20E-01	1.20E-01
S2	8.33E-02	2.50E-01	S7	3.38E-02	8.99E-01	S3	1.35E-01	8.57E-01	S8	1.16E-01	2.36E-01
S3	8.33E-02	3.34E-01	s10	3.19E-02	9.31E-01	S4	9.02E-02	9.47E-01	S14	1.14E-01	3.50E-01
S4	8.33E-02	4.17E-01	S9	2.98E-02	9.61E-01	S2	4.51E-02	9.92E-01	S7	1.05E-01	4.55E-01
S5	8.33E-02	5.00E-01	S2	2.89E-02	9.89E-01	S6	6.34E-03	9.99E-01	S9	8.55E-02	5.41E-01
S6	8.33E-02	5.84E-01	S8	6.51E-03	9.96E-01	S7	7.05E-04	9.99E-01	S12	7.71E-02	6.18E-01
S7	8.33E-02	6.67E-01	S11	2.13E-03	9.98E-01	S5	3.52E-04	1.00E+00	S2	7.06E-02	6.88E-01
S8	8.33E-02	7.50E-01	S3	1.86E-03	1.00E+00	S9	1.76E-04	1.00E+00	s10	6.61E-02	7.54E-01
S9	8.33E-02	8.33E-01	S12	1.42E-04	1.00E+00	s10	1.10E-05	1.00E+00	S1	6.04E-02	8.15E-01
s10	8.33E-02	9.17E-01	S6	6.67E-05	1.00E+00	S12	7.74E-06	1.00E+00	S15	3.12E-02	8.46E-01
S11	8.33E-02	1.00E+00	S4	6.21E-05	1.00E+00	S8	5.51E-06	1.00E+00	S16	2.75E-02	8.73E-01
			S13	2.03E-05	1.00E+00	S15	1.12E-06	1.00E+00	S17	1.93E-02	8.93E-01
			S5	4.14E-06	1.00E+00	S13	3.44E-07	1.00E+00	S19	1.92E-02	9.12E-01
			S14	1.58E-06	1.00E+00	S17	2.90E-07	1.00E+00	S4	1.69E-02	9.29E-01
			S15	6.60E-08	1.00E+00	S11	1.72E-07	1.00E+00	S6	1.47E-02	9.44E-01
			S16	5.11E-08	1.00E+00	S19	3.51E-08	1.00E+00	S20	1.27E-02	9.56E-01
						S14	2.15E-08	1.00E+00	S13	1.25E-02	9.69E-01
						S16	5.38E-09	1.00E+00	S11	1.09E-02	9.80E-01
						S18	2.69E-09	1.00E+00	S3	1.05E-02	9.90E-01
									S5	1.02E-02	1.00E+00
count	11		count	16		count	19		count	20	
25%	2		25%	1		25%	1		25%	2	
50%	5		50%	1		50%	1		50%	5	
90%	9		90%	2		90%	2		90%	12	

Table 6.3: FSM benchmark steady state probability analysis: planet, pma, s832 and s1494.

	planet			pma			s832			s1494	
states	prob.	accum.	states	prob.	accum.	states	prob.	accum.	states	prob.	accum.
S2	5.39E-02	5.39E-02	S7	1.52E-01	1.52E-01	S1	6.40E-01	6.40E-01	S1	8.10E-01	8.10E-01
S7	4.98E-02	1.04E-01	S3	1.37E-01	2.89E-01	S17	2.13E-01	8.53E-01	S15	1.01E-01	9.11E-01
S16	4.98E-02	1.54E-01	S1	1.14E-01	4.03E-01	S18	1.07E-01	9.60E-01	S25	3.80E-02	9.49E-01
S17	4.98E-02	2.03E-01	S4	1.14E-01	5.17E-01	S15	2.00E-02	9.80E-01	S21	1.66E-02	9.66E-01
S18	4.98E-02	2.53E-01	S6	9.49E-02	6.12E-01	S16	1.00E-02	9.90E-01	S17	1.27E-02	9.78E-01
S3	4.15E-02	2.95E-01	S2	9.11E-02	7.03E-01	S3	4.83E-03	9.95E-01	S20	8.31E-03	9.87E-01
S8	4.05E-02	3.35E-01	S11	8.69E-02	7.90E-01	S2	3.79E-03	9.99E-01	S5	4.31E-03	9.91E-01
S9	4.05E-02	3.76E-01	S5	5.69E-02	8.47E-01	S5	7.32E-04	9.99E-01	S34	2.53E-03	9.93E-01
S26	3.73E-02	4.13E-01	S8	3.80E-02	8.85E-01	S4	6.44E-04	1.00E+00	S24	1.75E-03	9.95E-01
S27	3.73E-02	4.50E-01	S21	2.85E-02	9.13E-01	S6	3.05E-06	1.00E+00	S8	1.26E-03	9.96E-01
S4	3.58E-02	4.86E-01	S22	2.85E-02	9.42E-01	S7	2.03E-07	1.00E+00	S13	8.76E-04	9.97E-01
S5	3.13E-02	5.17E-01	s10	2.28E-02	9.65E-01	S19	8.48E-09	1.00E+00	S44	6.32E-04	9.98E-01
S6	3.13E-02	5.49E-01	S9	1.52E-02	9.80E-01	S12	3.69E-09	1.00E+00	S16	3.16E-04	9.98E-01
s10	2.96E-02	5.78E-01	S12	1.45E-02	9.94E-01	S8	1.70E-09	1.00E+00	S36	2.88E-04	9.99E-01
S12	2.96E-02	6.08E-01	S23	3.56E-03	9.98E-01	S24	1.70E-09	1.00E+00	S18	2.19E-04	9.99E-01
S14	2.96E-02	6.37E-01	S13	1.81E-03	1.00E+00	S13	4.69E-10	1.00E+00	S28	2.19E-04	9.99E-01
S19	2.80E-02	6.65E-01	S14	4.52E-04	1.00E+00	S25	1.39E-10	1.00E+00	S23	1.49E-04	9.99E-01
S11	2.02E-02	6.86E-01	S15	4.52E-04	1.00E+00	S14	1.25E-10	1.00E+00	S39	1.09E-04	9.99E-01
S13	2.02E-02	7.06E-01	S17	2.37E-04	1.00E+00	S20	6.02E-11	1.00E+00	S42	1.09E-04	9.99E-01
S15	2.02E-02	7.26E-01	S24	2.22E-04	1.00E+00	S9	5.65E-11	1.00E+00	S30	5.47E-05	9.99E-01
S31	1.87E-02	7.45E-01	S18	3.16E-05	1.00E+00	S23	9.25E-12	1.00E+00	S32	5.47E-05	9.99E-01
S32	1.87E-02	7.63E-01	S16	1.48E-05	1.00E+00	s10	3.77E-12	1.00E+00	S47	5.47E-05	1.00E+00
S43	1.85E-02	7.82E-01	S19	1.98E-06	1.00E+00	S22	6.17E-13	1.00E+00	S38	3.72E-05	1.00E+00
S44	1.85E-02	8.00E-01	S20	1.24E-07	1.00E+00	S21	2.69E-13	1.00E+00	S45	2.79E-05	1.00E+00
S45	1.85E-02	8.19E-01				S11	2.51E-13	1.00E+00	S22	2.74E-05	1.00E+00
S46	1.85E-02	8.37E-01							S2	1.37E-05	1.00E+00
S20	1.49E-02	8.52E-01							S4	3.85E-06	1.00E+00
S28	1.40E-02	8.66E-01							S27	1.38E-06	1.00E+00
S1	1.35E-02	8.80E-01							S14	1.29E-06	1.00E+00
S25	1.12E-02	8.91E-01							S33	1.04E-06	1.00E+00
S33	1.07E-02	9.02E-01							S6	4.89E-07	1.00E+00
S29	9.34E-03	9.11E-01							s10	3.45E-07	1.00E+00
S30	9.34E-03	9.20E-01							S41	2.60E-07	1.00E+00
S36	9.34E-03	9.30E-01							S31	1.97E-07	1.00E+00
S37	9.34E-03	9.39E-01							S19	1.30E-07	1.00E+00
S38	9.34E-03	9.48E-01							S46	1.30E-07	1.00E+00
S42	8.71E-03	9.57E-01							S40	1.22E-07	1.00E+00
S21	7.47E-03	9.65E-01							S29	8.62E-08	1.00E+00
S47	6.22E-03	9.71E-01							S7	6.51E-08	1.00E+00
S34	5.34E-03	9.76E-01							S3	3.70E-08	1.00E+00
S39	4.98E-03	9.81E-01							S11	3.36E-08	1.00E+00
S35	4.00E-03	9.85E-01							S35	2.16E-08	1.00E+00
S22	3.73E-03	9.89E-01							S43	2.16E-08	1.00E+00
S48	3.11E-03	9.92E-01							S37	9.97E-09	1.00E+00
S41	2.49E-03	9.94E-01							S48	8.40E-09	1.00E+00
S23	1.87E-03	9.96E-01							S12	4.99E-09	1.00E+00
S24	1.87E-03	9.98E-01							S9	3.15E-09	1.00E+00
S40	1.87E-03	1.00E+00							S26	2.49E-09	1.00E+00
count	48		count	24		count	25		count	48	
25%	5		25%	2		25%	1		25%	1	
50%	12		50%	4		50%	1		50%	1	
90%	30		90%	9		90%	2		90%	1	

Table 6.4: FSM benchmark steady state probability analysis: sand, styr and tbk.

	sand			styr			tbk	
states	prob.	accum.	states	prob.	accum.	states	prob.	accum.
S1	1.04E-01	1.04E-01	S1	6.38E-01	6.38E-01	S1	3.42E-01	3.42E-01
S6	9.90E-02	2.03E-01	S13	1.66E-01	8.04E-01	S17	3.42E-01	6.84E-01
S9	7.01E-02	2.73E-01	S11	9.45E-02	8.99E-01	S14	6.56E-02	7.50E-01
S32	6.19E-02	3.35E-01	S12	2.10E-02	9.20E-01	S30	6.56E-02	8.15E-01
S30	5.57E-02	3.91E-01	S2	2.04E-02	9.40E-01	S15	1.16E-02	8.27E-01
S31	5.57E-02	4.46E-01	s10	1.60E-02	9.56E-01	S16	1.16E-02	8.38E-01
S28	4.95E-02	4.96E-01	S3	1.45E-02	9.70E-01	S31	1.16E-02	8.50E-01
S29	4.95E-02	5.45E-01	S14	1.28E-02	9.83E-01	S32	1.16E-02	8.62E-01
S8	4.38E-02	5.89E-01	S9	1.07E-02	9.94E-01	S2	5.79E-03	8.67E-01
S26	4.33E-02	6.33E-01	S15	3.49E-03	9.97E-01	S3	5.79E-03	8.73E-01
S27	4.33E-02	6.76E-01	S4	6.60E-04	9.98E-01	S4	5.79E-03	8.79E-01
S24	3.71E-02	7.13E-01	S29	3.47E-04	9.98E-01	S5	5.79E-03	8.85E-01
S25	3.71E-02	7.50E-01	S8	3.30E-04	9.99E-01	S6	5.79E-03	8.91E-01
S22	3.09E-02	7.81E-01	S16	2.18E-04	9.99E-01	S7	5.79E-03	8.96E-01
S23	3.09E-02	8.12E-01	S17	1.96E-04	9.99E-01	S8	5.79E-03	9.02E-01
S20	2.48E-02	8.37E-01	S23	1.25E-04	9.99E-01	S9	5.79E-03	9.08E-01
S21	2.48E-02	8.61E-01	S7	3.07E-05	9.99E-01	s10	5.79E-03	9.14E-01
S18	1.86E-02	8.80E-01	S5	2.16E-05	9.99E-01	S11	5.79E-03	9.20E-01
S19	1.86E-02	8.99E-01	S6	2.06E-05	9.99E-01	S12	5.79E-03	9.25E-01
S2	1.84E-02	9.17E-01	S18	1.96E-05	9.99E-01	S13	5.79E-03	9.31E-01
S7	1.65E-02	9.34E-01	S30	1.64E-05	9.99E-01	S18	5.79E-03	9.37E-01
S16	1.24E-02	9.46E-01	S20	1.36E-05	9.99E-01	S19	5.79E-03	9.43E-01
S17	1.24E-02	9.58E-01	S24	4.31E-06	9.99E-01	S20	5.79E-03	9.48E-01
S11	8.25E-03	9.67E-01	S26	4.04E-06	9.99E-01	S21	5.79E-03	9.54E-01
S13	8.25E-03	9.75E-01	S22	2.18E-06	9.99E-01	S22	5.79E-03	9.60E-01
S14	6.19E-03	9.81E-01	S21	1.36E-06	9.99E-01	S23	5.79E-03	9.66E-01
S15	6.19E-03	9.87E-01	S19	1.23E-06	9.99E-01	S24	5.79E-03	9.72E-01
S3	4.80E-03	9.92E-01	S28	1.54E-07	9.99E-01	S25	5.79E-03	9.77E-01
s10	4.38E-03	9.96E-01	S27	1.39E-07	9.99E-01	S26	5.79E-03	9.83E-01
S12	2.06E-03	9.98E-01	S25	1.35E-07	9.99E-01	S27	5.79E-03	9.89E-01
S4	7.07E-04	9.99E-01				S28	5.79E-03	9.95E-01
S5	7.07E-04	1.00E+00				S29	5.79E-03	1.00E+00
count	32		count	30		count	32	
25%	3		25%	1		25%	1	
50%	8		50%	1		50%	2	
90%	19		90%	3		90%	14	

scenario which means when a fault hits a memory element, it is protected. We assume that combinational logic implementing the next state equations of memory elements are non shared.

- TMRing both combinational logic and memory elements: at which each memory element (aka FF) is triplicated along with its combinational logic as shown in Figure 6.1 and then a voter circuit is added. The combinational logic is assumed to be non shared between memory elements.

Each of the experiments in each circuit follows the framework of Figure 6.2. The FSM or state table - formatted in kiss format- either taken as is or after adding redundant states are converted to *blif* using sis. At that stage the combinational logic of the state variables equations are shared one time to get the original sequential circuit implementation, and not shared another time to see the impact of logic separation on the reliability of the circuit. After that, blif file of the sequential circuit is converted to bench format which is accepted by hope tool. The bench files, either taken as original circuits or protected by our method, are accepted as an argument by our evaluator described in Section 5.2. Ultimately, failure rate versus the number of injected faults for each circuit is plotted using Matlab.

6.2 Calculating the Area Overhead

Techniques involving adding redundancy to enhance reliability often pay the price in terms of area. We calculate the size of sequential circuits relative to the

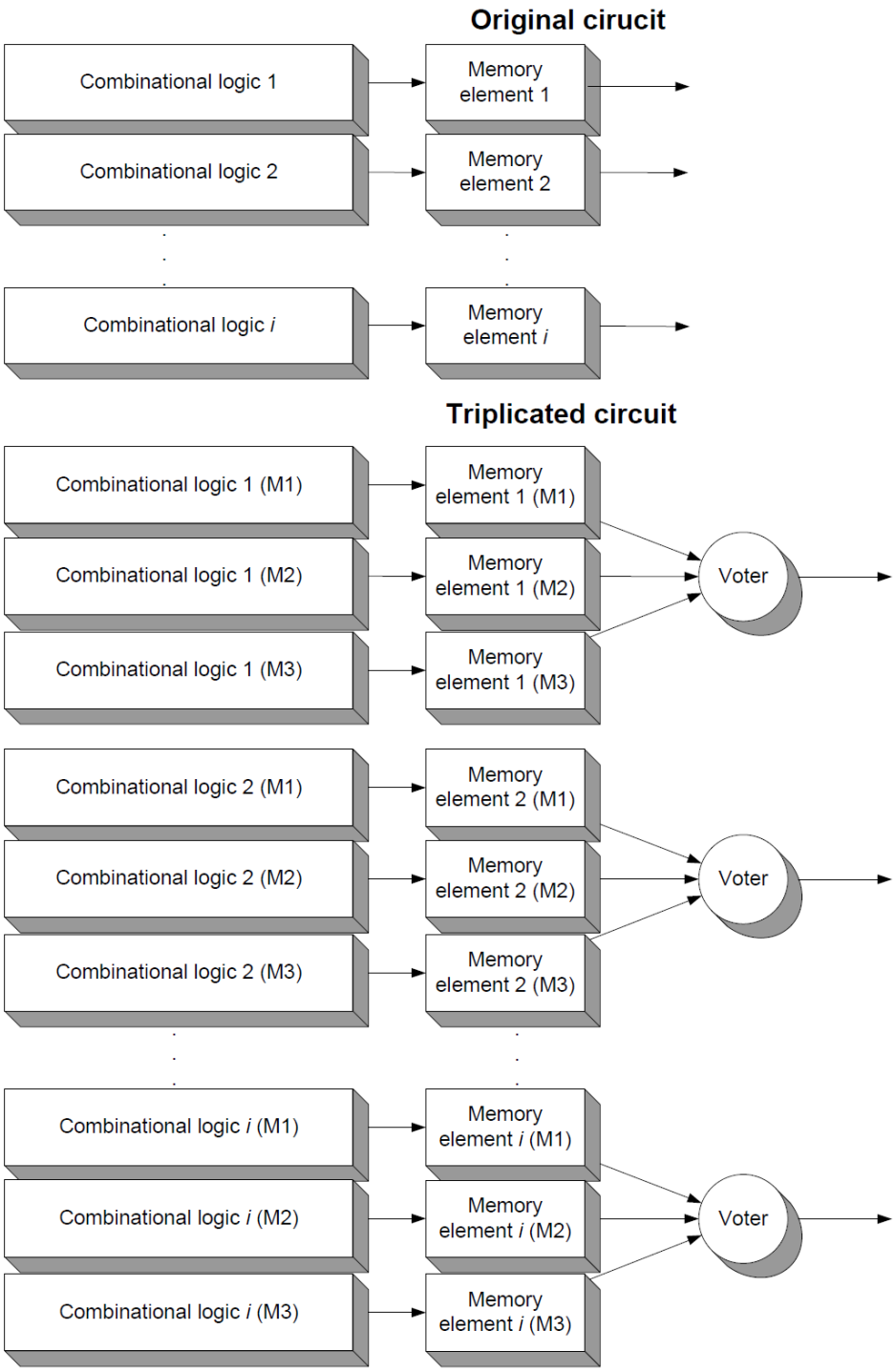


Figure 6.1: TMRing both combinational logic and memory elements.

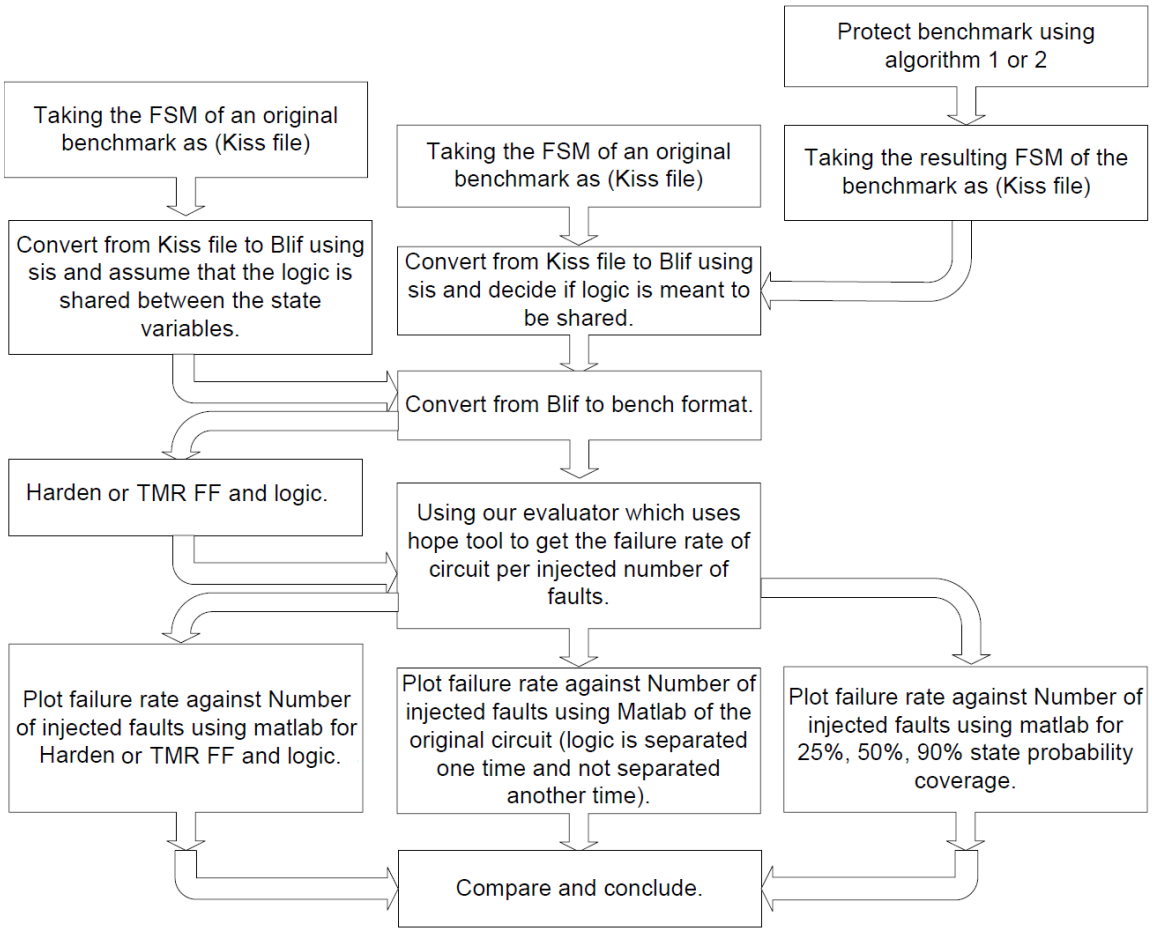


Figure 6.2: Framework of experiments.

Table 6.5: Size of gates.

Gate	Size
NOT	1
NAND2	2
NAND3	3
NAND4	4
NOR2	2
NOR3	3
NOR4	4
D-FF	12
Harden D-FF	30

size of a single inverter. Table 6.5 shows the library that we used to estimate the size of circuits.

Particularly in the hardening case, we assume that the size of each latch is 2.5 times the size of an original latch. This assumption is concluded from the paper in [38].

6.3 Algorithm 1 Results

In this section, we will apply Algorithm 1 to the circuits stated in Section 6.1. The resulting findings are discussed particularity for dk14 and for the remaining circuits in the next few sections.

6.3.1 Case Study: dk14 Benchmark

The sequential circuit dk14, which has 7 states, is picked to discuss general results. State 3 is protected to achieve a minimum of 25% state probability cov-

erage; states 3, 1 and 2 are protected to achieve a minimum of 50% coverage; and states 3, 2, 1 and 5 are protected to achieve around 90% coverage. Figure 6.3 shows 25%, 50%, 90% and 100% state probability coverage failure rate versus the number of injected faults. It is compared to the original circuit with and without sharing the logic between state variables. Also, it is compared to hardening and TMRing both the logic and the flip flops.

Results shown in Figure 6.3 show that as we cover more number of states the failure rate drops down and hence the reliability is increased. This is clearly shown by comparing the 25%, 50%, 90% and 100% state probability coverage failure rates. It is particularly marked that 50% and 90% give a significant drop in failure rate. In Section 3.4, the correlation between failure rate and state probability is throughly discussed. According to results in Figure 6.3, consistent observations are found. The failure rate decreases as more states with higher probability of occurrence are protected. Also, it is noticeable that by separating the logic to ensure that all single errors happening in the protected states are tolerated gives a lower failure rate than the original circuit with sharing the logic. This is expected since SET can cause multiple errors in different memory elements in the original circuit with shared logic.

Hardening techniques and TMRing have failure rate curves which are better than the original dk14 circuit and its non-shared version. However, non sharing the logic has stronger impact in lowering the failure rate compared to the original shared circuit. By comparing our method to hardening and TMRing methods,

they give a very comparable measurement to protecting states that yield a minimum of 25% state probability coverage especially when the number of injected faults increases beyond 15 faults.

To complete the picture, Figure 6.4 compares the area of the implementation of the discussed curves. It is shown that the area of the implemented circuit increases as the number of protected states, providing 25%, 50%, 90% up to 100% state probability coverage, increases. This is a logical expected price paid to redeem the pros of enhancing the reliability. However, at 100% state probability coverage the area overhead is unacceptably large even though the failure rate drop is the best. To avoid this excessive area overhead, protecting 50% or 90% can be chosen at which a significant drop in terms of failure rate compared to original circuit is obtained and the area overhead is significantly reduced.

Furthermore, one interesting observation is found for dk14 when protecting states 3 and 1 is compared against protecting states 3 and 2. Since the probability calculation for each pair of states, i.e. (state3, state1) and (state3, state2), are nearly the same, it is expected that the resulting failure rate curves be very close to each other. However, as it is shown in Figure 6.5, we observe that protecting states 3 and 2 is better in terms of failure rate than protecting states 3 and 1.

By analyzing their failure rate, protection rate as well as masking rate (See Figures 6.5, 6.6 and 6.7), we found that even though the protection rate of the pair (state3,state1) is more than the protection rate of (state3,state2), the masking rate of (state3,state2) is more than the masking rate of (state3,state1). This masking

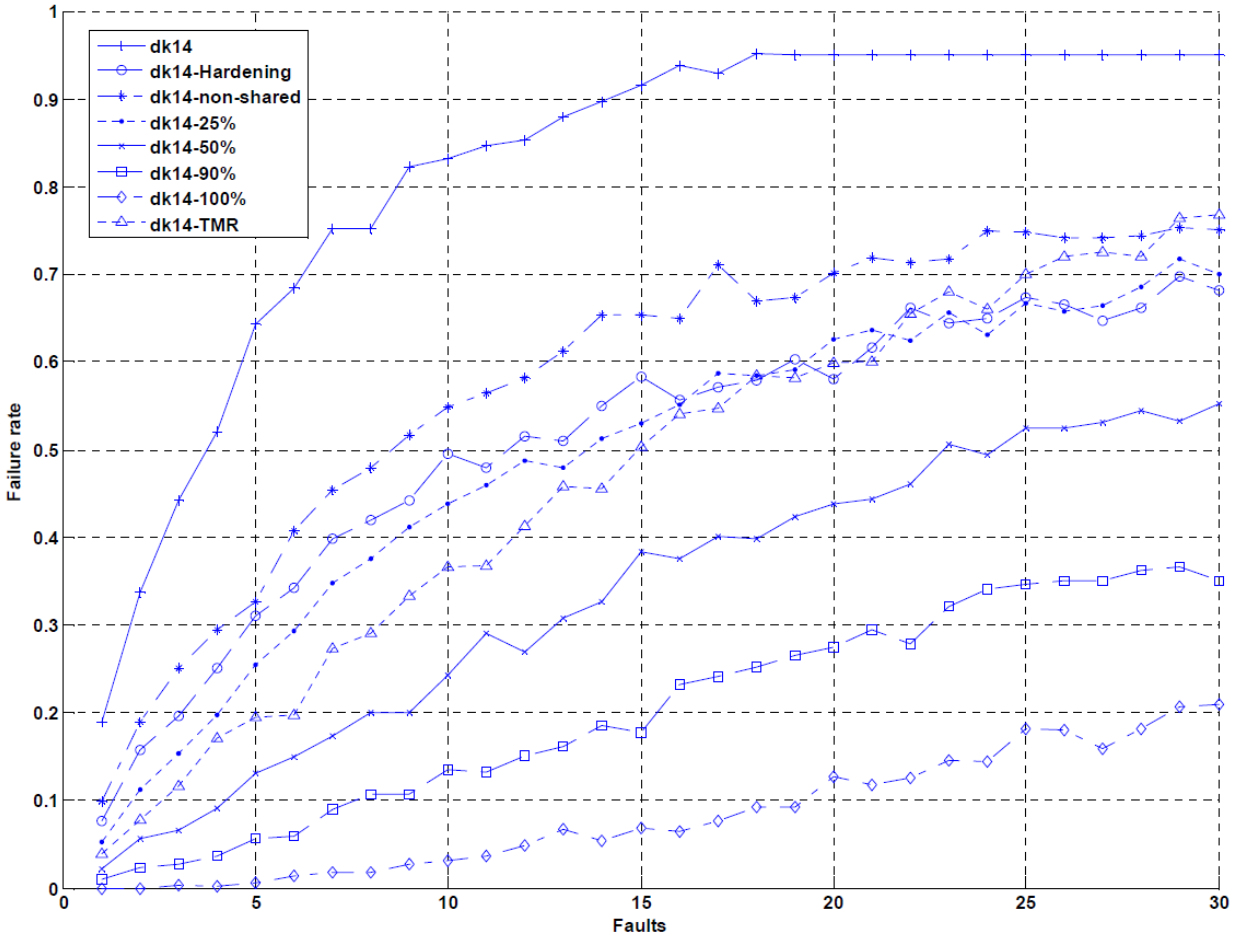


Figure 6.3: Failure rate vs Faults for dk14.

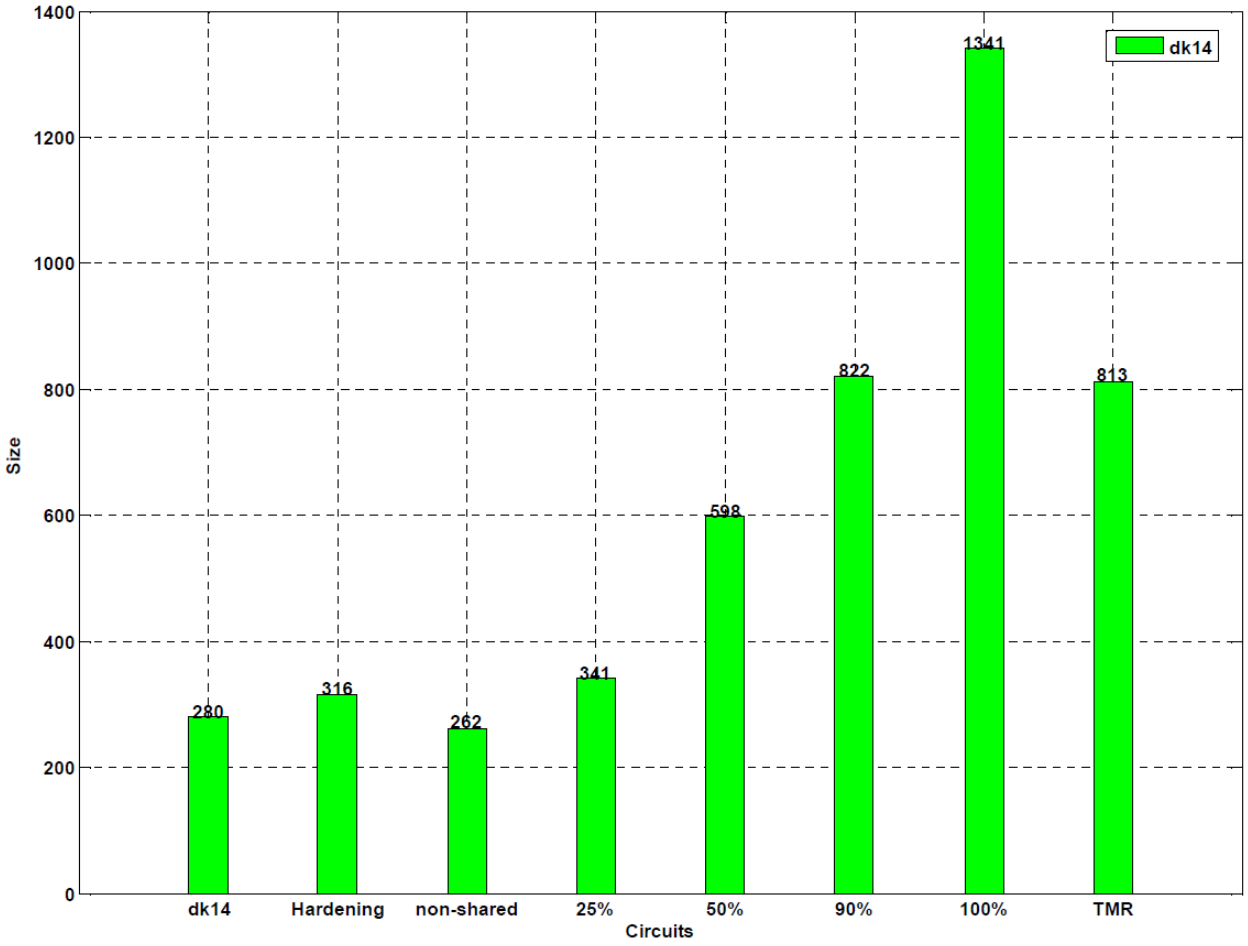


Figure 6.4: Size of dk14 experiments.

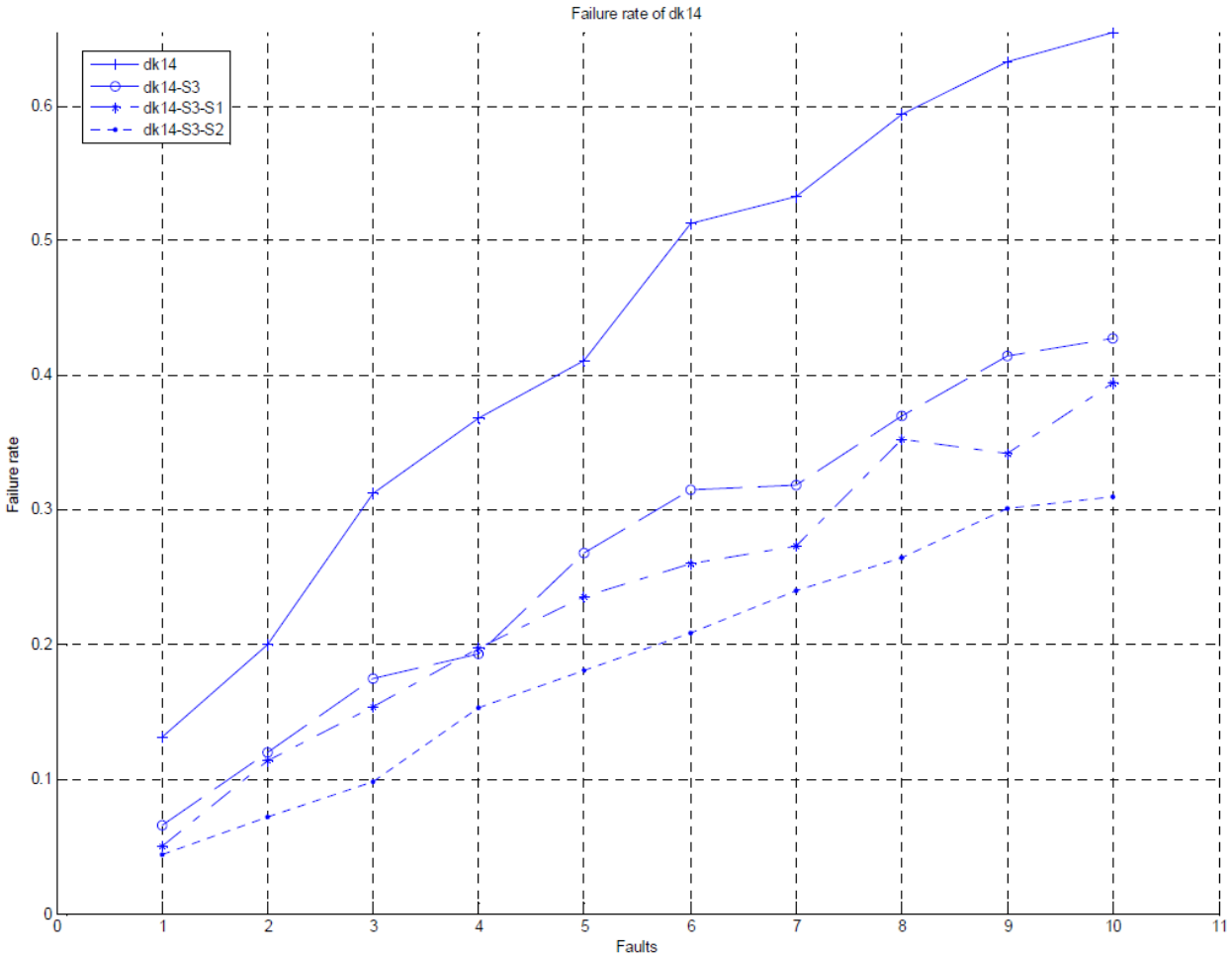


Figure 6.5: Failure rate of protecting state3-state1 vs state3-state2.

is the dominate factor that overcome the protection impact using redundant states against SET when state3 and state2 are protected. In other words, the masking rate of the resulting circuit when state3 and state2 are protected causes the failure rate to drop more than the one when state 3 and state 1 are protected.

6.3.2 Other Benchmarks Results

We can classify sequential circuits based on their states probability into three classes. The *first class* contains states which are close to each other in terms

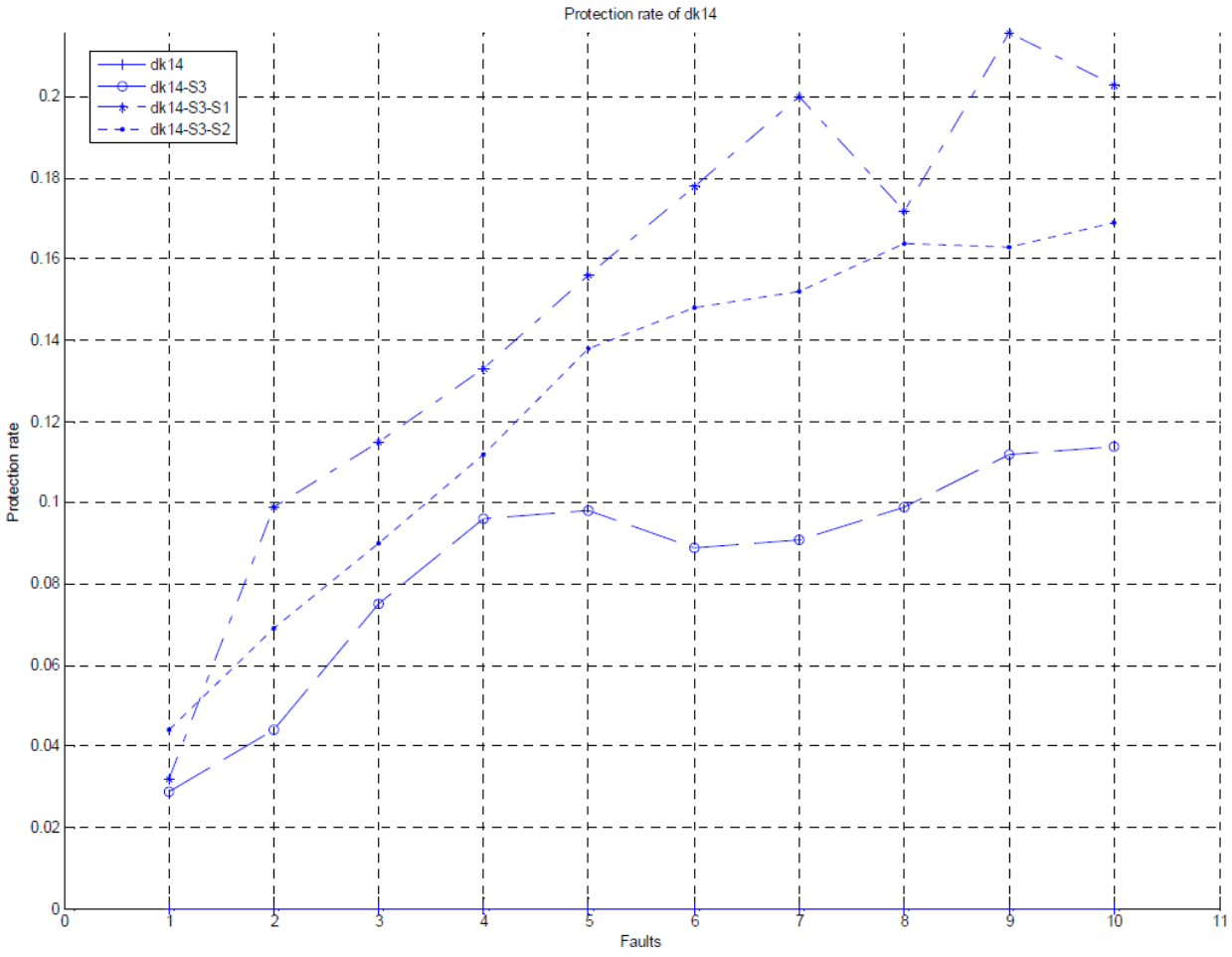


Figure 6.6: Protection rate of protecting state3-state1 vs state3-state2.

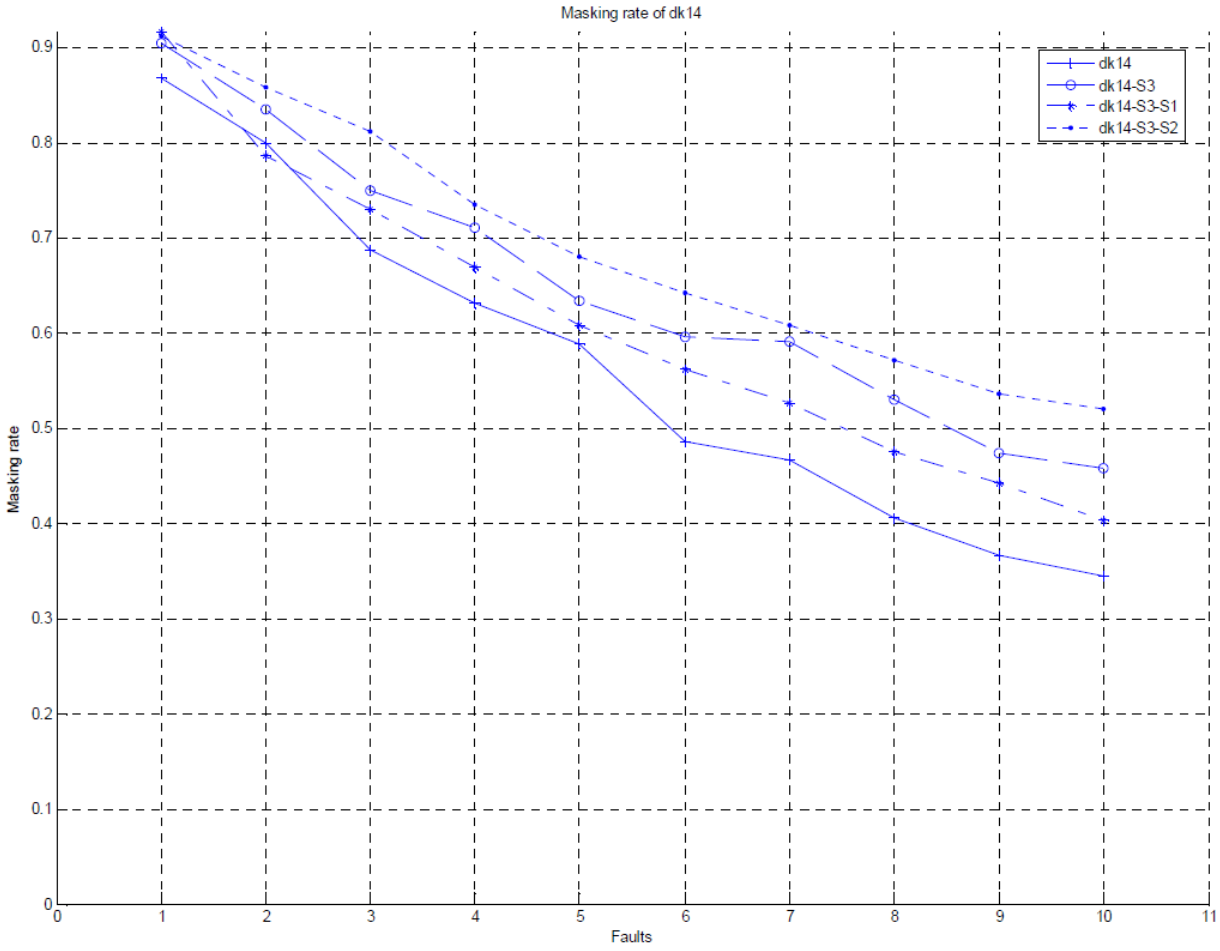


Figure 6.7: Masking rate of protecting state3-state1 vs state3-state2.

of steady state probability. Hence, many number of states (sometimes up to half of them) are protected to obtain a minimum of 25%, 50% and 90% state probability coverage. The *second class* contains states which have significant difference from each other in terms of steady state probability. Hence, few number of states (sometimes 1 or 2 states) are protected to obtain a minimum of 25%, 50% and 90% state probability coverage. The *third class* contains states which have similar steady state probability. Hence, the percentage of the number of protected states to obtain a certain state probability coverage is equal to the specified state probability coverage. The resulting number of protected states in the *third class* is more than the ones in the *first class*.

The sequential circuit dk14 which is discussed in the previous section is one of the *first class* circuits. Other circuits such as bbara, bbsse, s1, pma and sand behave like dk14 circuit. Their simulation results illustrating failure rate and area comparison are given in Figures 6.8 to 6.17. The results show that as we cover more number of states, the failure rate drops down and hence the reliability is increased. This is clearly shown by comparing the 25%, 50%, 90% and 100% state probability coverage failure rates compared to original circuit with shared logic. The area overhead increased as the number of protected states increases which is reflected by the increase in state probability coverage percentage. The same scenario is observed when they are compared to the the non-shared version of the original circuit except that the non-shared version gives lower failure rate measurement than the shared version but a very comparable area measurement.

By comparing with hardening technique, protecting states yielding a minimum of 25% state probability coverage gives better failure rate than hardening in cost of slightly more area overhead. TMRing gives failure rate curve which is between the failure rate curve of protecting states yielding a minimum of 25% state probability coverage and 50% state probability coverage. However, the area overhead of TMRing is more than protecting state with 50% state probability coverage with 0.5 up to 1 time additional area overhead. Our technique yielding 100% state probability coverage gives the best failure rate reduction compared to all other methods. However, the area overhead is excessively increased up to 9 times compared to the original circuit. A recommended solution with much lower area overhead and a comparable failure rate is protecting states that yield 90% state probability coverage.

Other circuits such as cse, keyb, styr and s832 fall under the *second class* and show different results than the *first class*. Their corresponding failure rate and sizes are shown in Figures 6.18 to 6.25, respectively. They involve protecting a very few number of states to get 50% and 90% state probability coverage. It is notable that by adding slightly more hardware to the original circuit in an intelligent way, such as when protecting states yielding a minimum of 50% or 90% probability coverage, we gain huge improvement in terms of failure rate. Comparing to hardening and TMRing methods, protecting states yielding 90% state probability coverage gives better results in terms of failure rate with area overhead very comparable to hardening and about 50% less than TMRing. This is mainly because

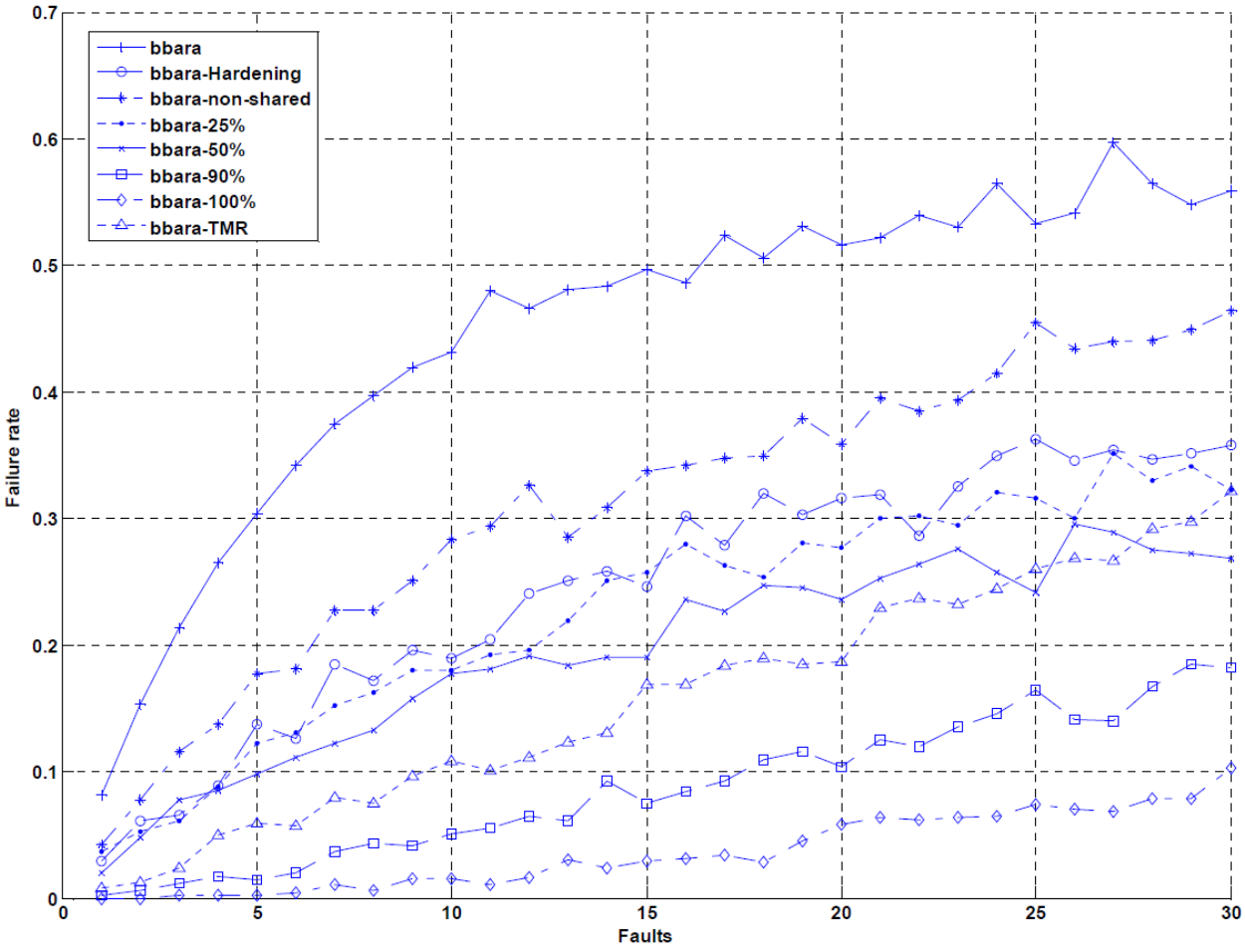


Figure 6.8: Failure rate vs Faults for bbara.

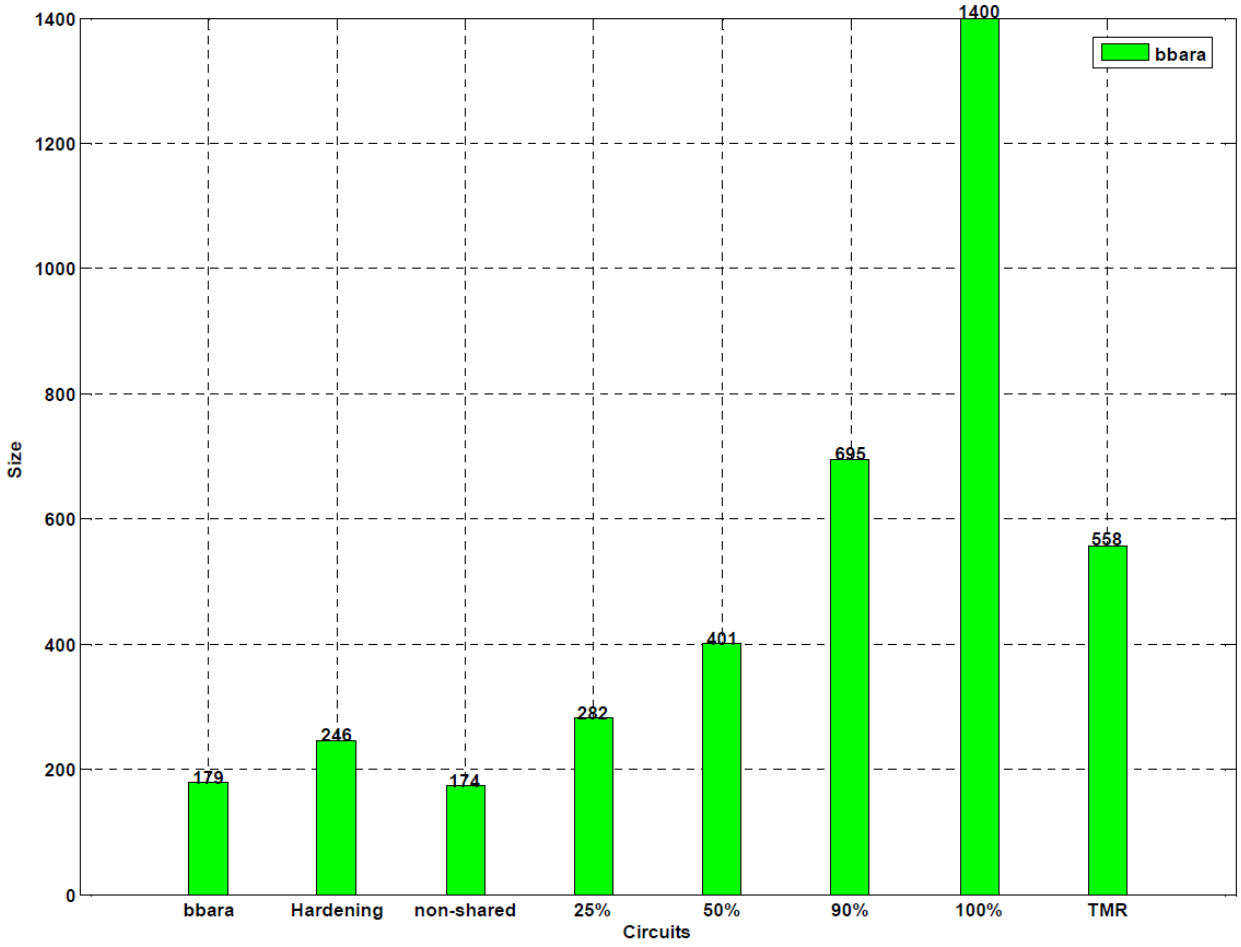


Figure 6.9: Size of bbara experiments.

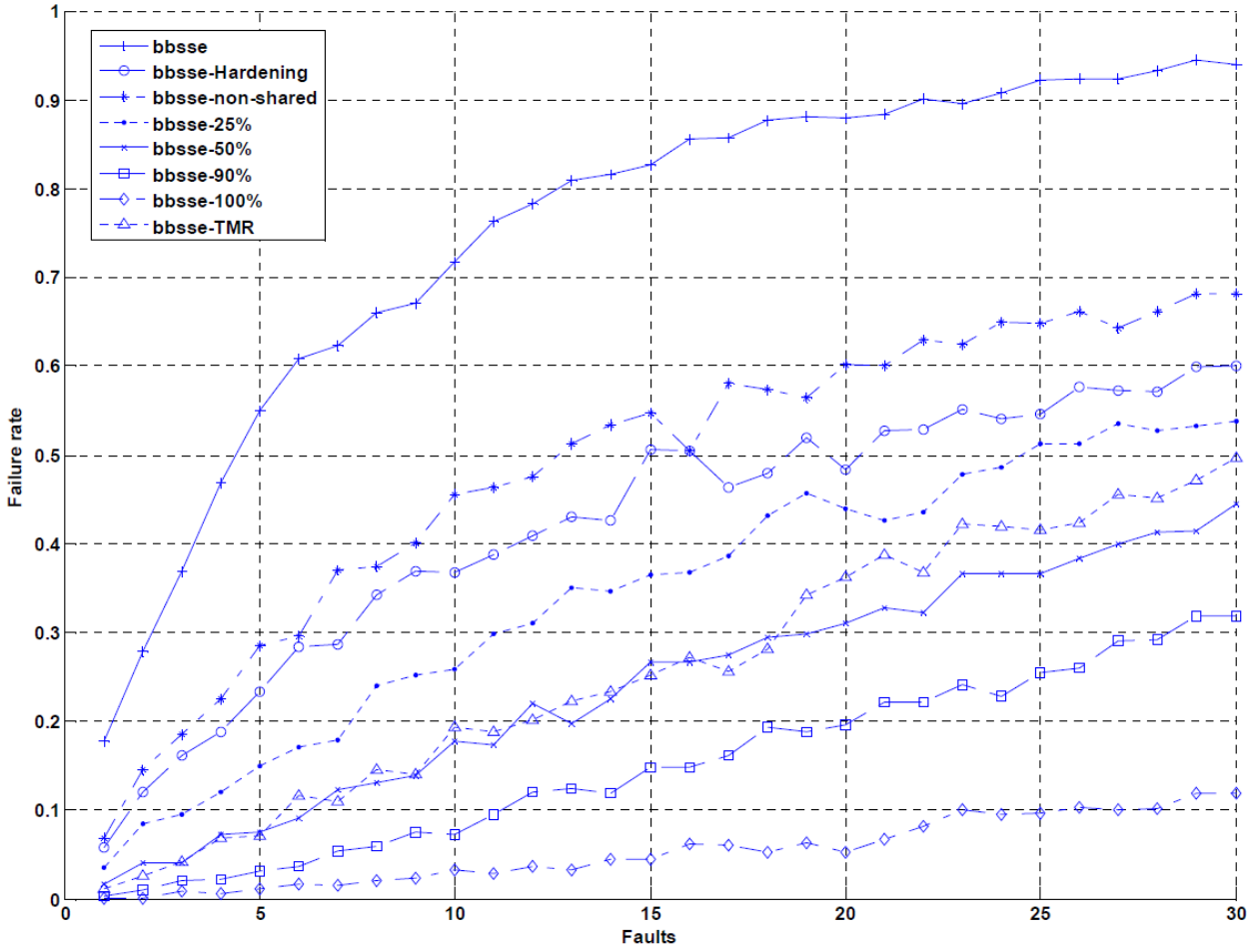


Figure 6.10: Failure rate vs Faults for bbsse.

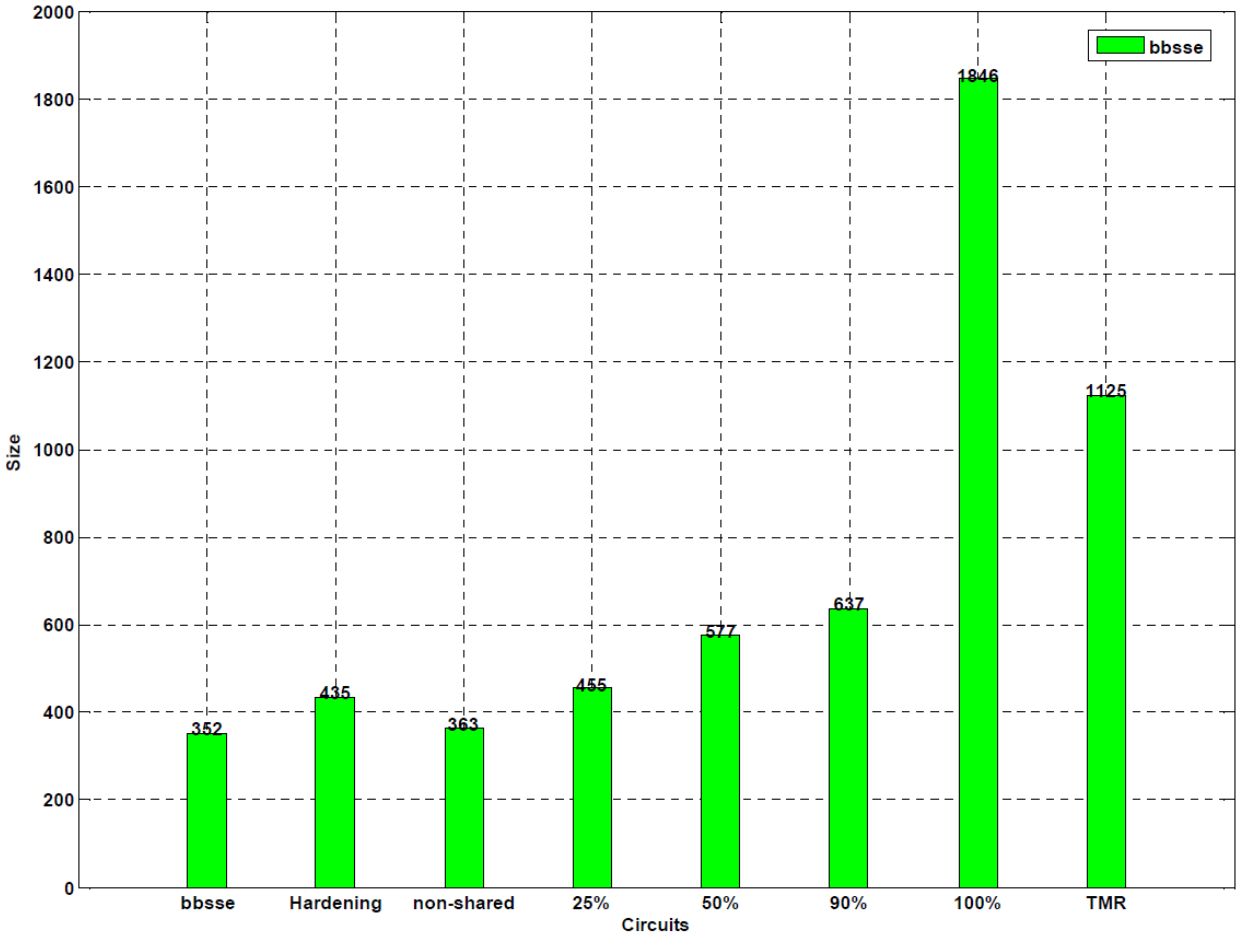


Figure 6.11: Size of bbsse experiments.

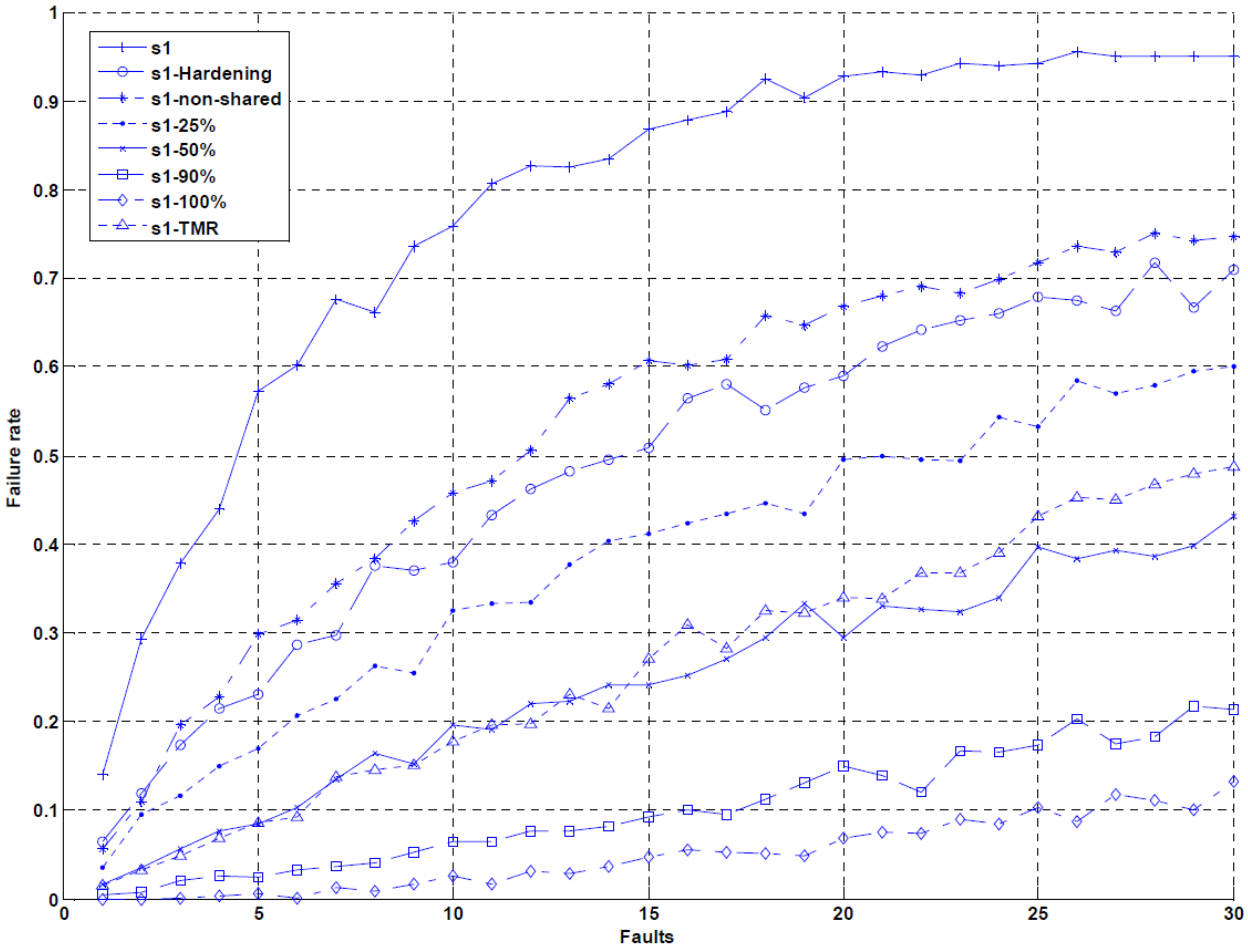


Figure 6.12: Failure rate vs Faults for s1.

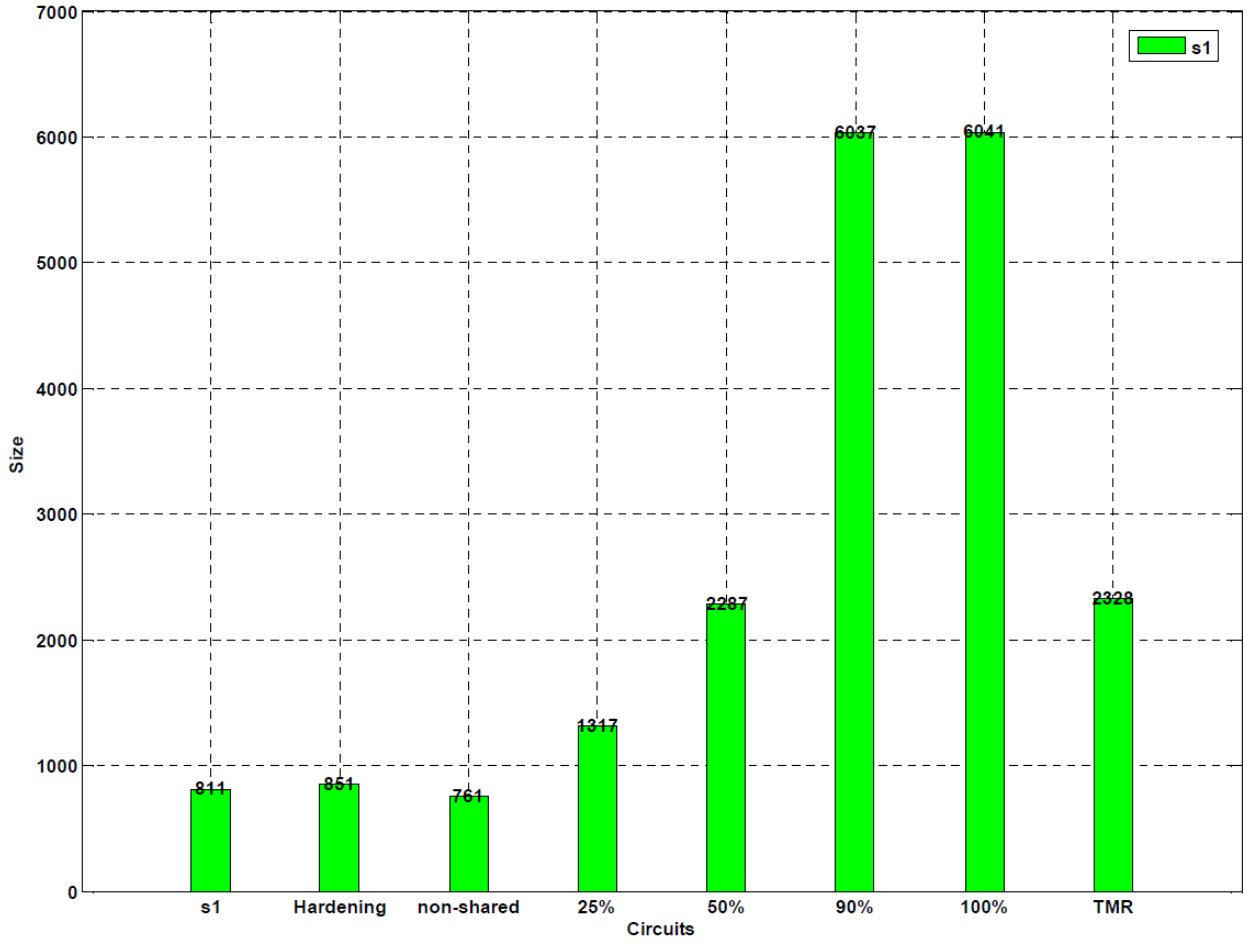


Figure 6.13: Size of s1 experiments.

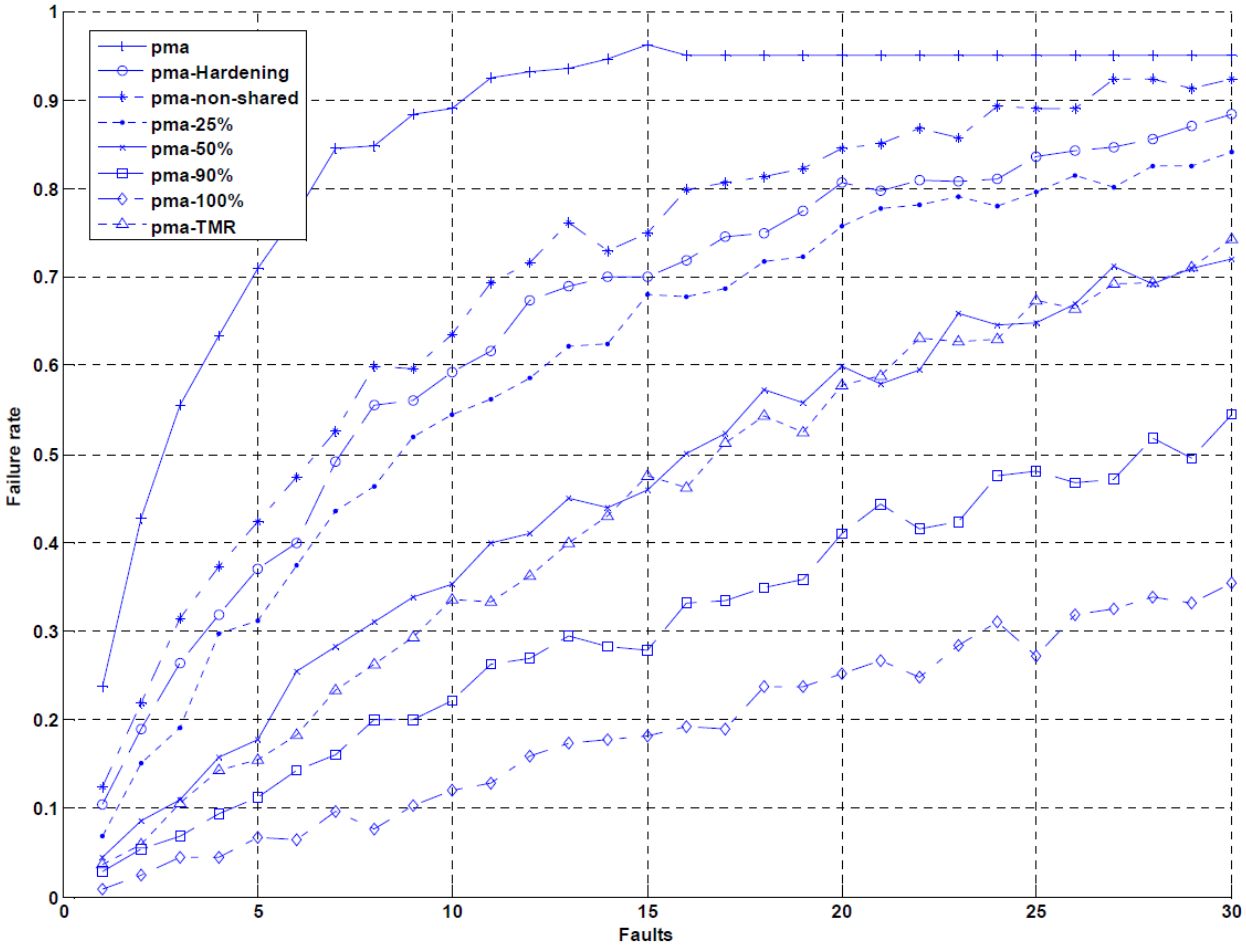


Figure 6.14: Failure rate vs Faults for pma.

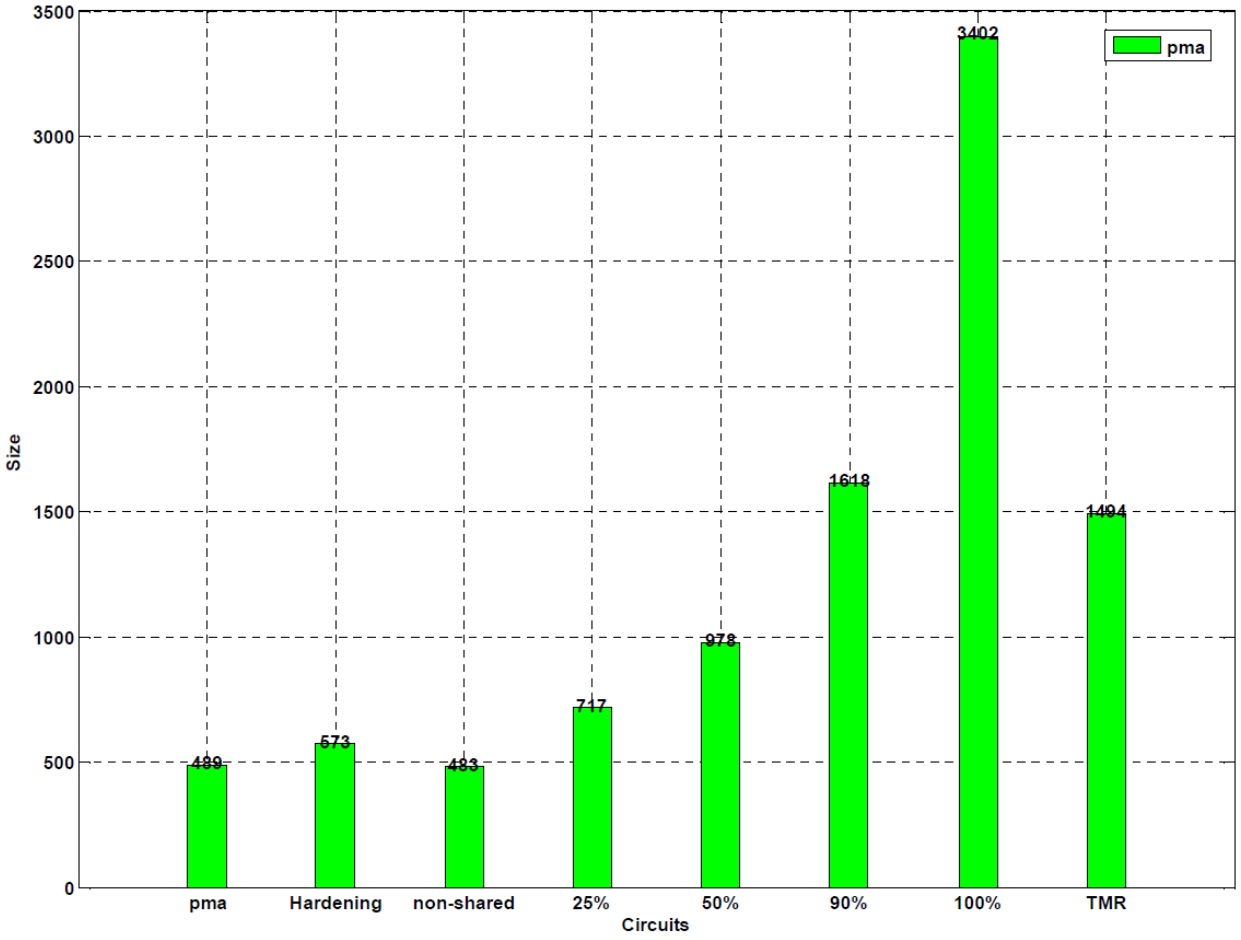


Figure 6.15: Size of pma experiments.

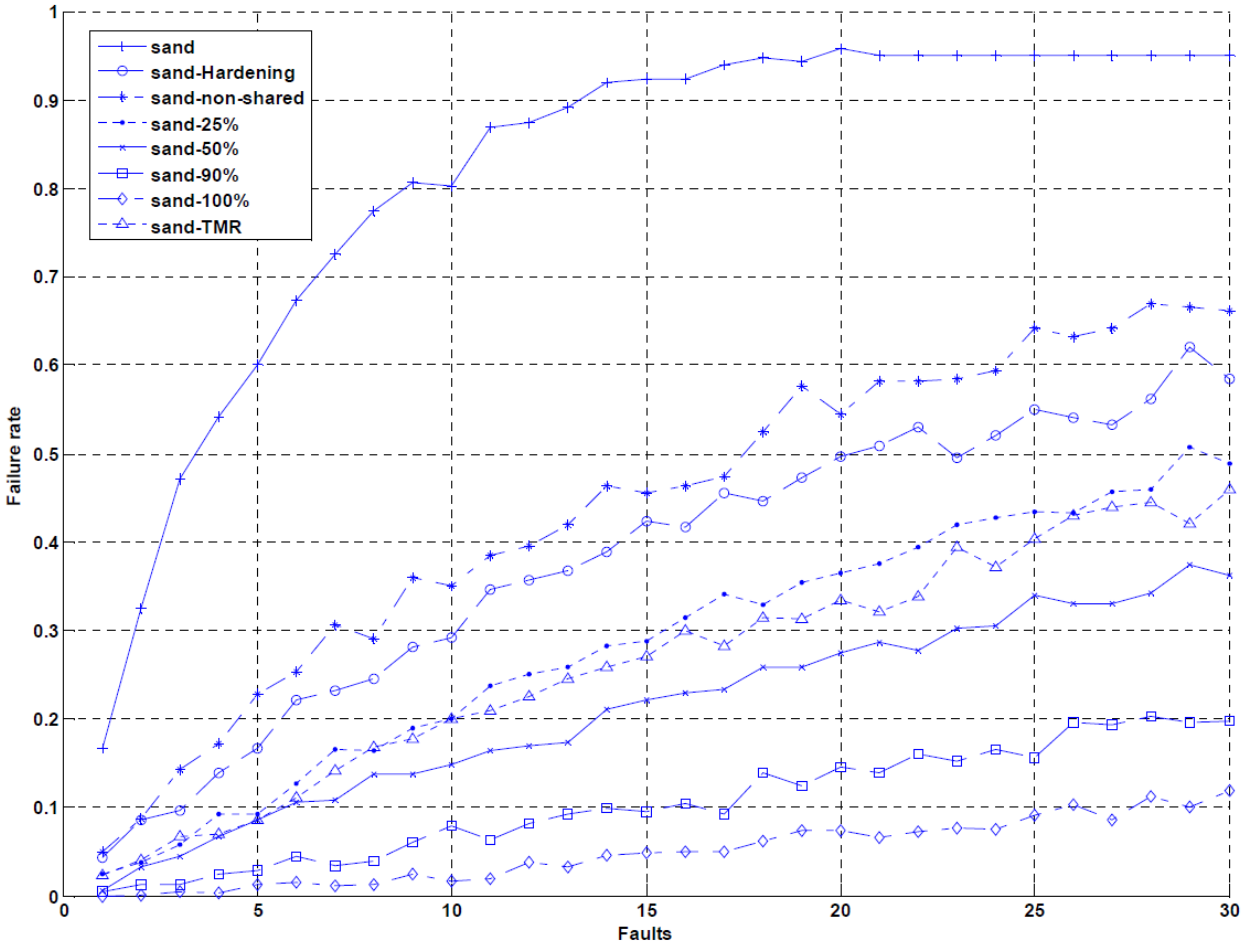


Figure 6.16: Failure rate vs Faults for sand.

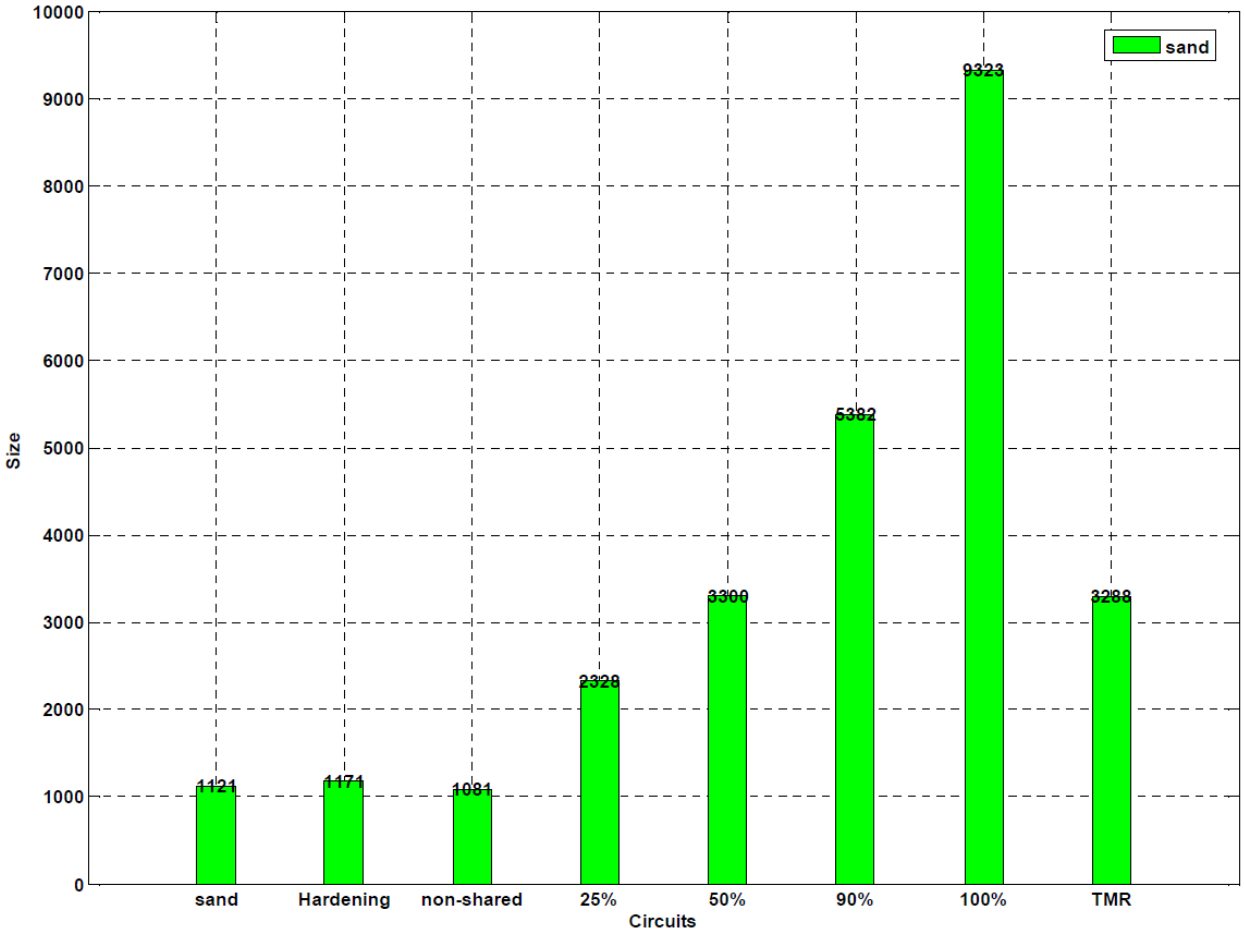


Figure 6.17: Size of sand experiments.

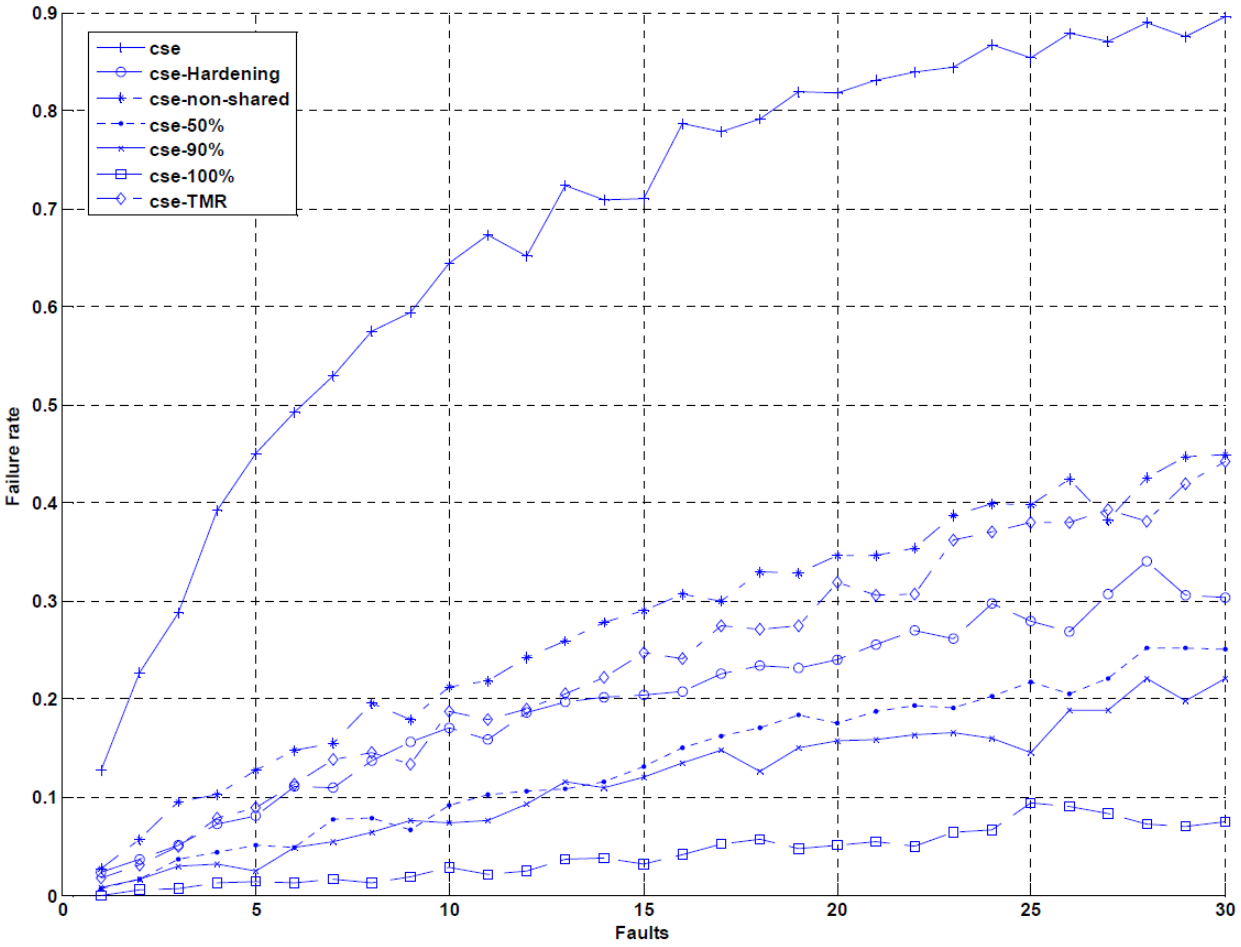


Figure 6.18: Failure rate vs Faults for cse.

hardening the memory elements alone doesn't give enough protection and TM-Ring fails if the faults happen in the voter circuit. As we mentioned in *first class* circuits, our technique yielding 100% state probability coverage gives the best failure rate reduction compared to all other methods. However, the area overhead is high. Hence, 90% state probability coverage protection is recommended.

If the steady state probabilities of the states in the FSM of sequential circuit are very far away from each other, our technique achieves significant improvement in terms of failure rate and area overhead with a very comparable area overhead to

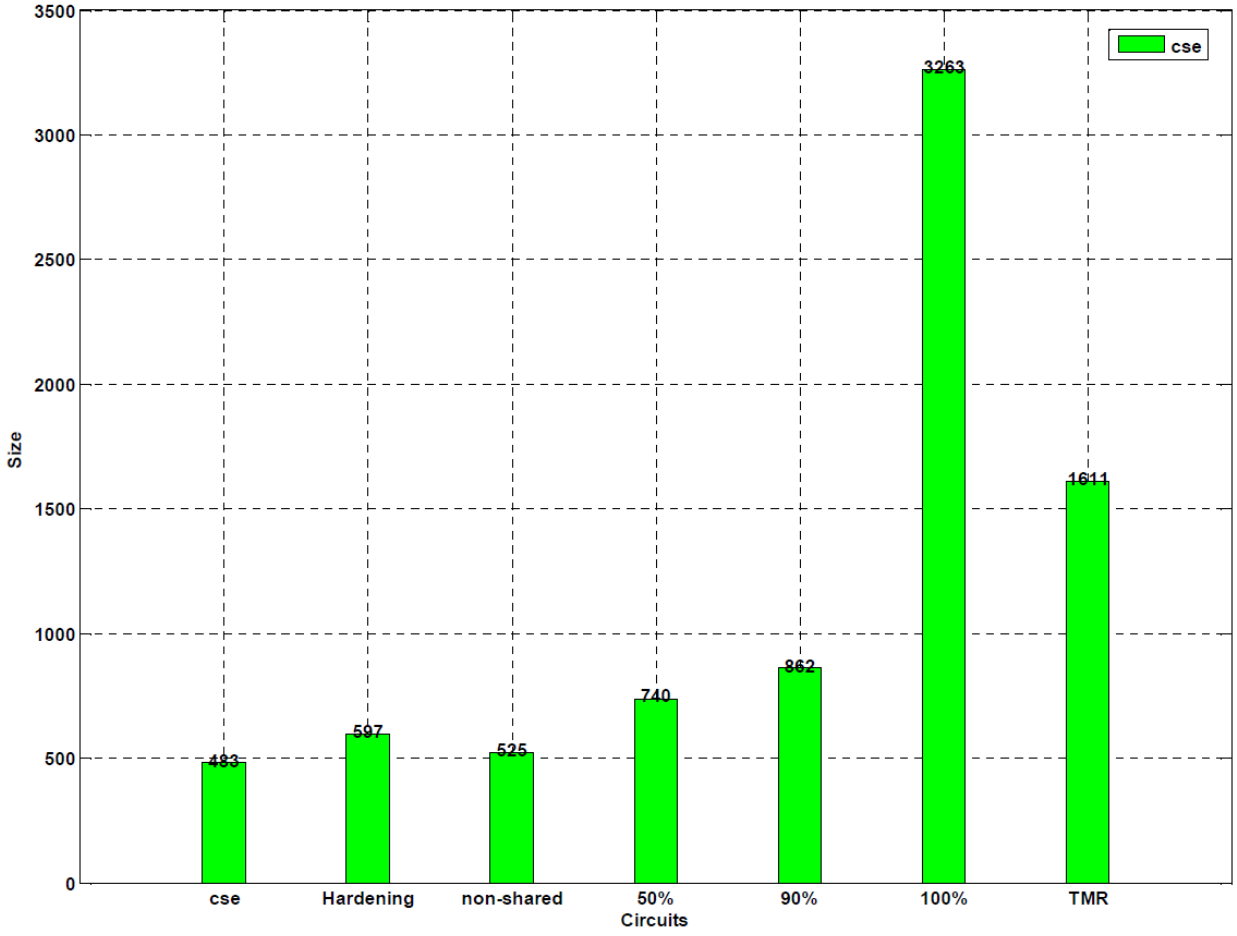


Figure 6.19: Size of cse experiments.

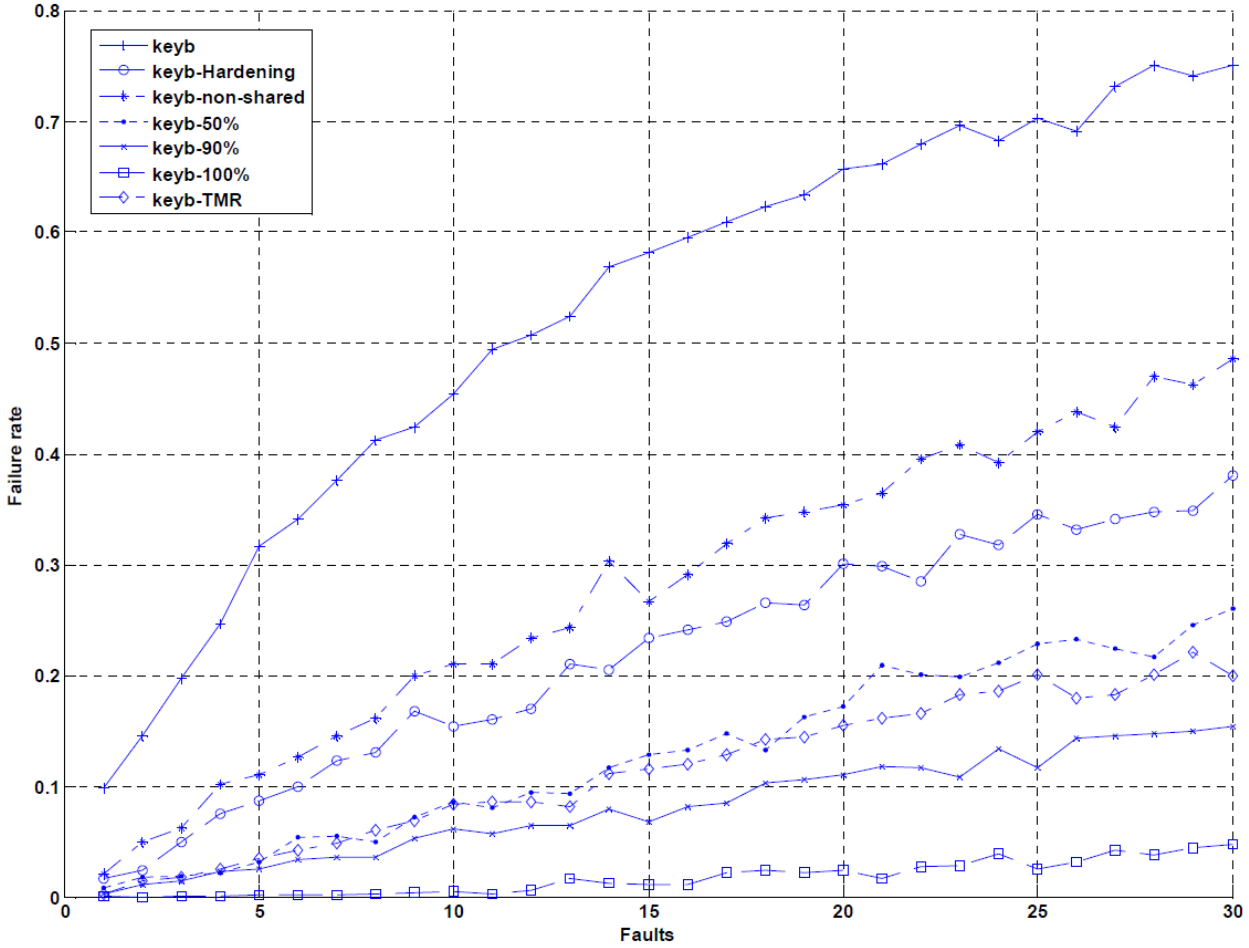


Figure 6.20: Failure rate vs Faults for keyb.

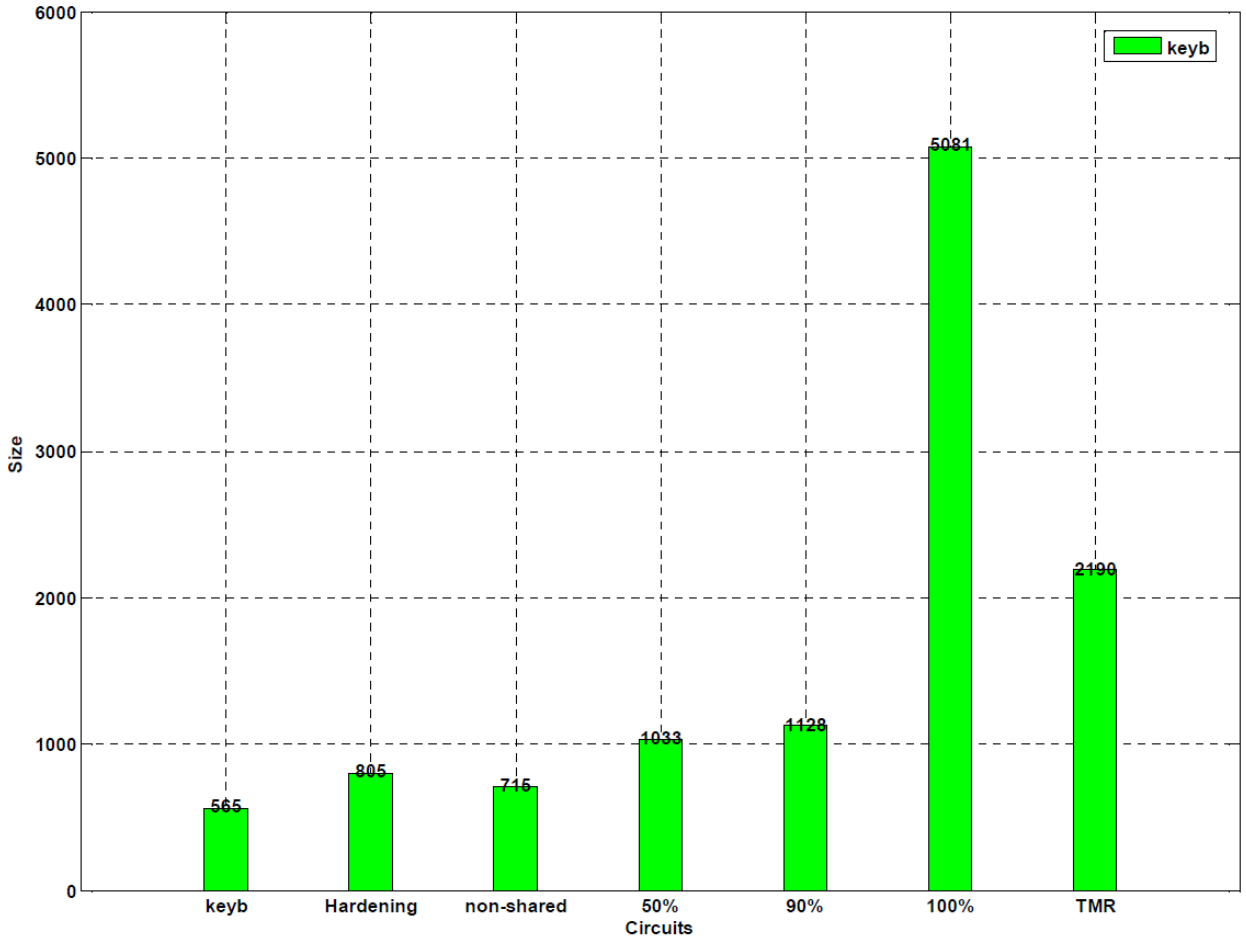


Figure 6.21: Size of keyb experiments.

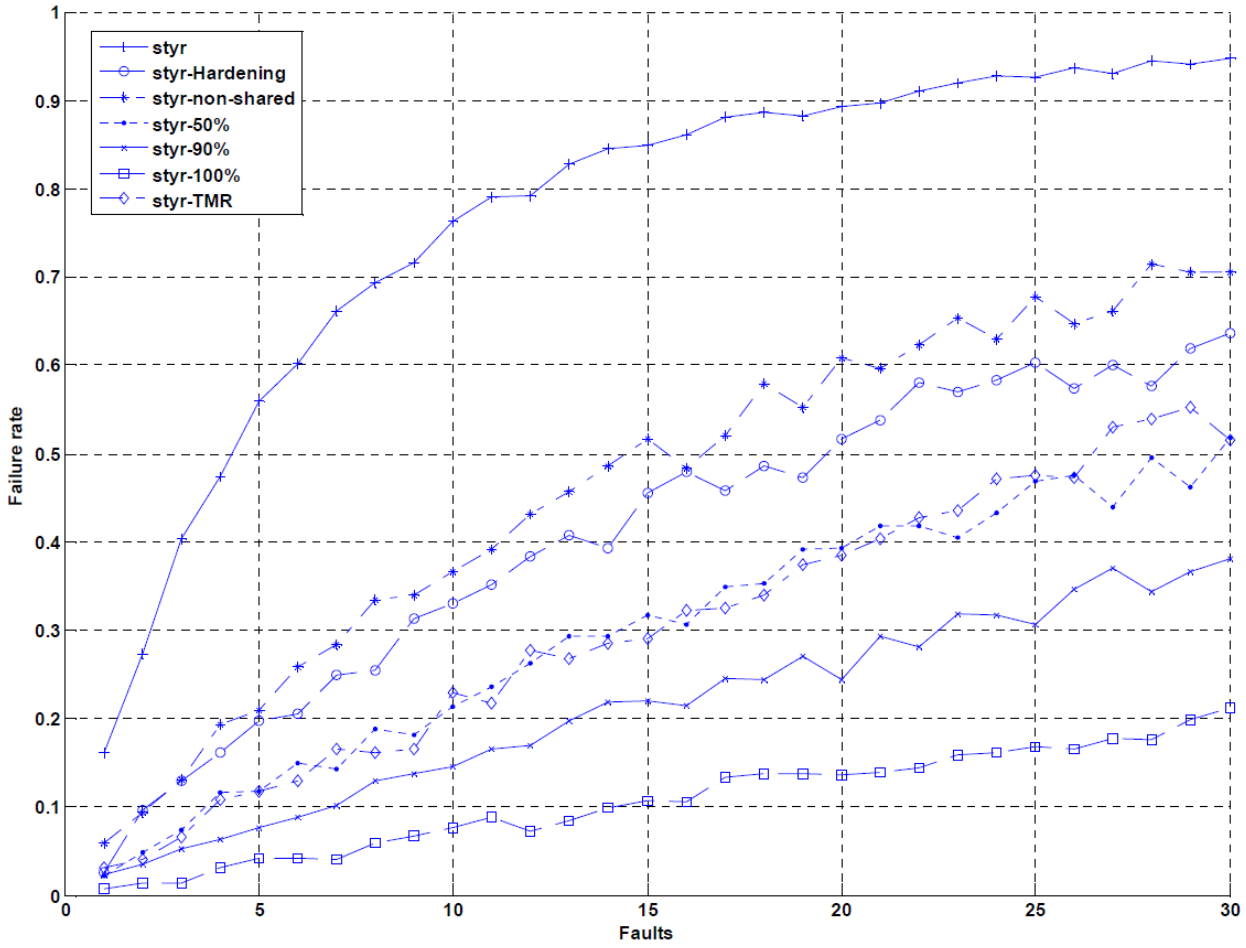


Figure 6.22: Failure rate vs Faults for styr.

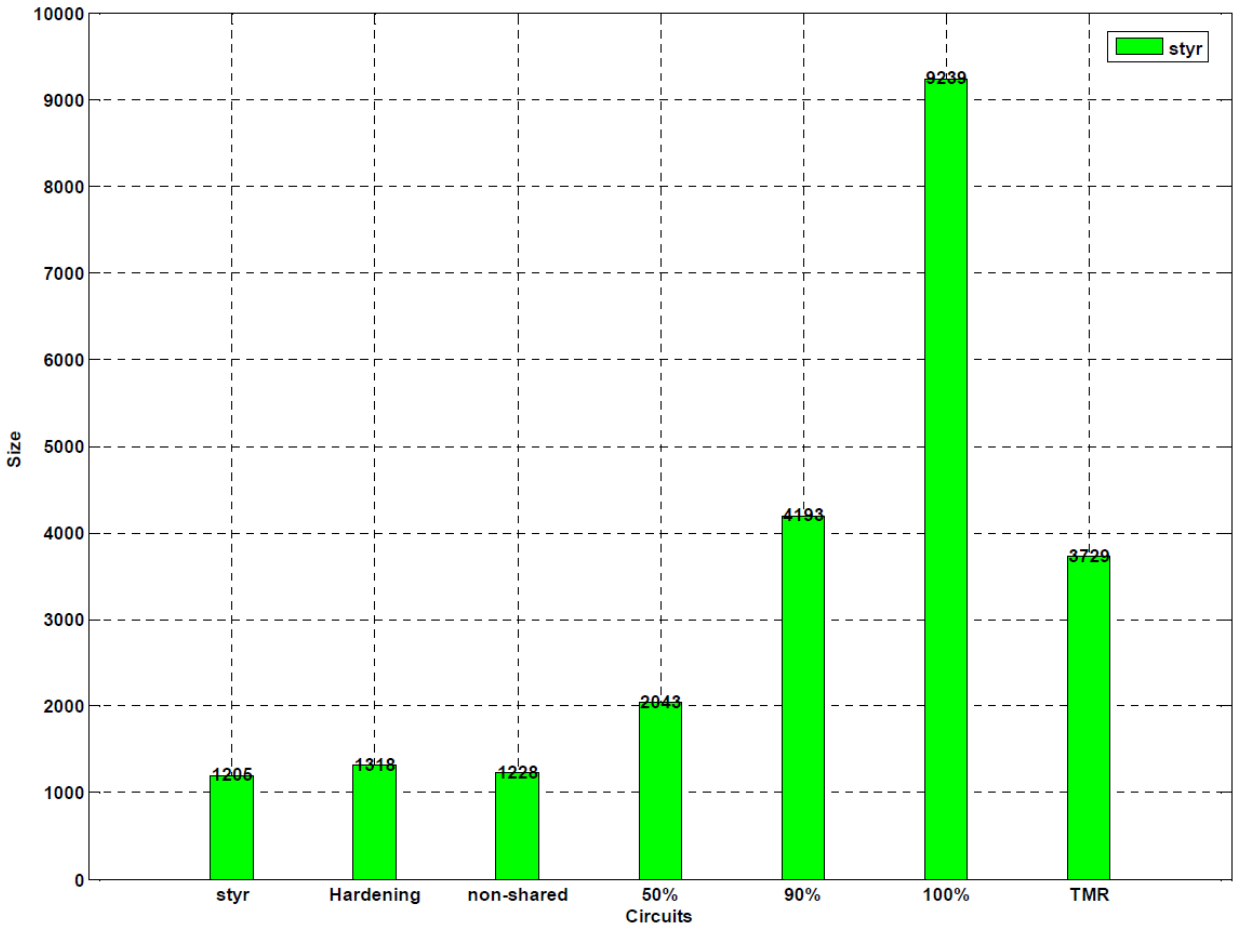


Figure 6.23: Size of styr experiments.

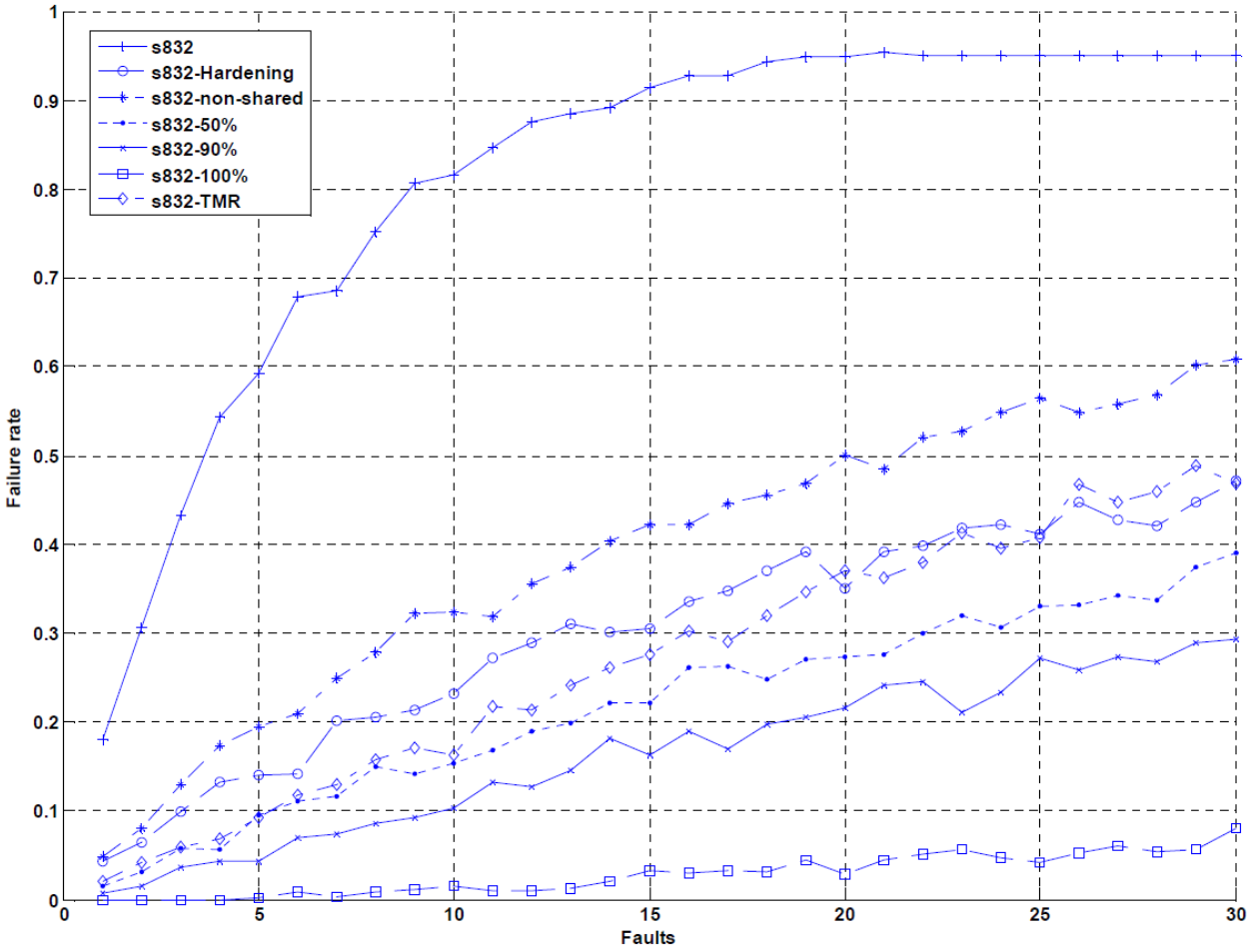


Figure 6.24: Failure rate vs Faults for s842.

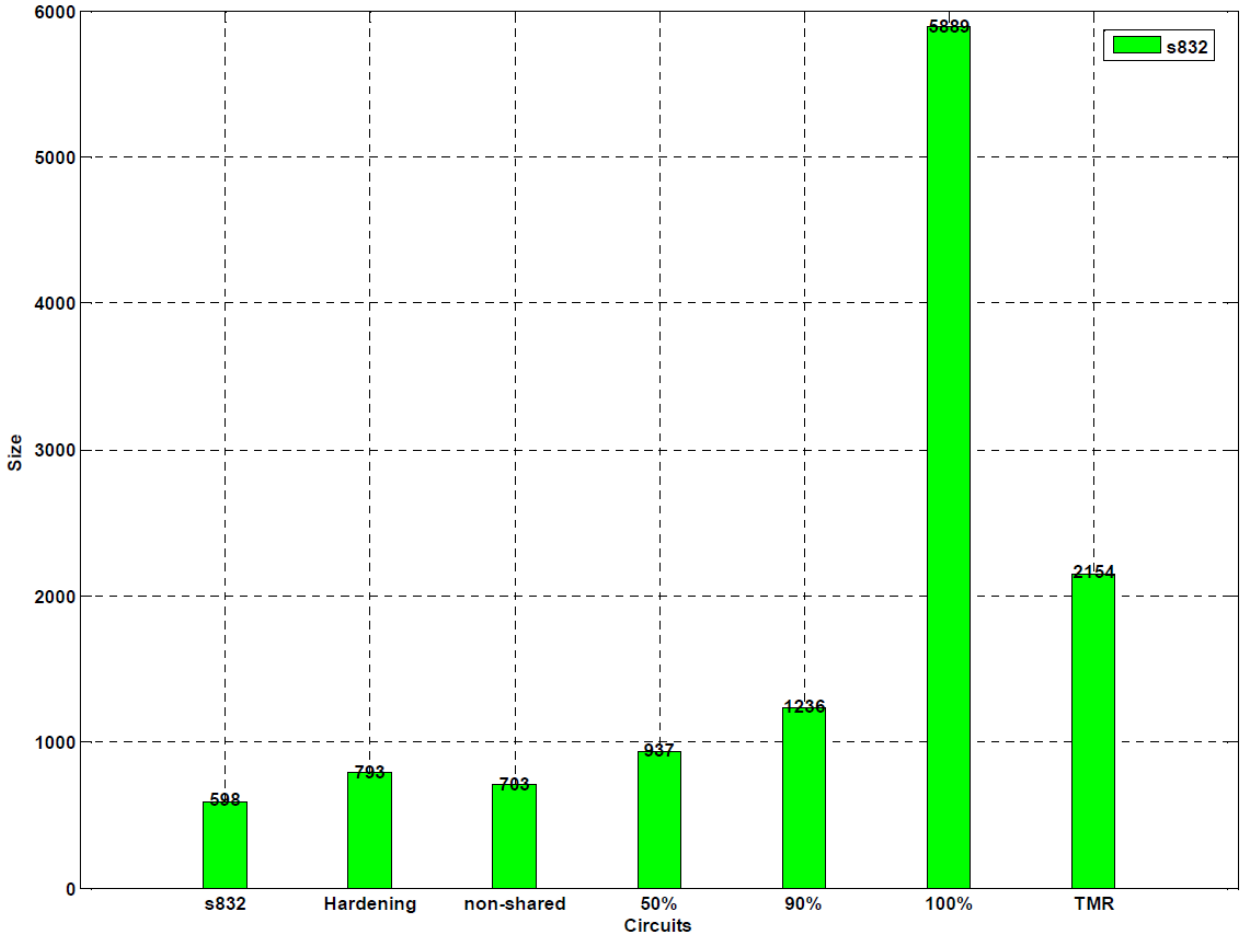


Figure 6.25: Size of s832 experiments.

the original circuit especially in cases such as covering 50% or even 90% of the state probability. This is because only few number of states are protected. However, if the steady state probabilities of the states are equal (i.e. *third class*) or very close to each other (i.e. *first class*), although our method results in a significant drop in terms of failure rate than the original circuit, it adds more area overhead to the circuit that can be sometimes more than TMRing both the combinational logic and the memory elements especially in 90% probability coverage case. To see that we can compare *second class* circuits such as cse, keyb, styr and s832 (See Figures 6.18 to 6.25) with *third class* circuits such as lion9 and train11 (See Figures 6.26 to 6.29). In train11 case, we observe that 90% state probability coverage protection is better than protecting all the states in terms of failure rate. The main reason for this behavior is that the masking in 90% state probability coverage protection case is more than the masking when all states are protected.

Two of the big FSM benchmark circuits in terms of states are planet (belongs to *first class*) and s1494 (belongs to *second class*). They both have 48 states in their FSM. The circuit planet involves protecting many states because the states probability are very close to each other. On the other hand, for the circuit s1494, protecting 1 state out of 48 achieves about 90% state probability coverage. In Figures 6.30 to 6.33, both circuits show better performance in terms of failure rate especially with 90% state probability coverage. Moreover, both circuits are better than hardening and TMRing methods. However, specifically in s1494 case, the powerfulness of our method is shown. The failure rate is hugely improved and

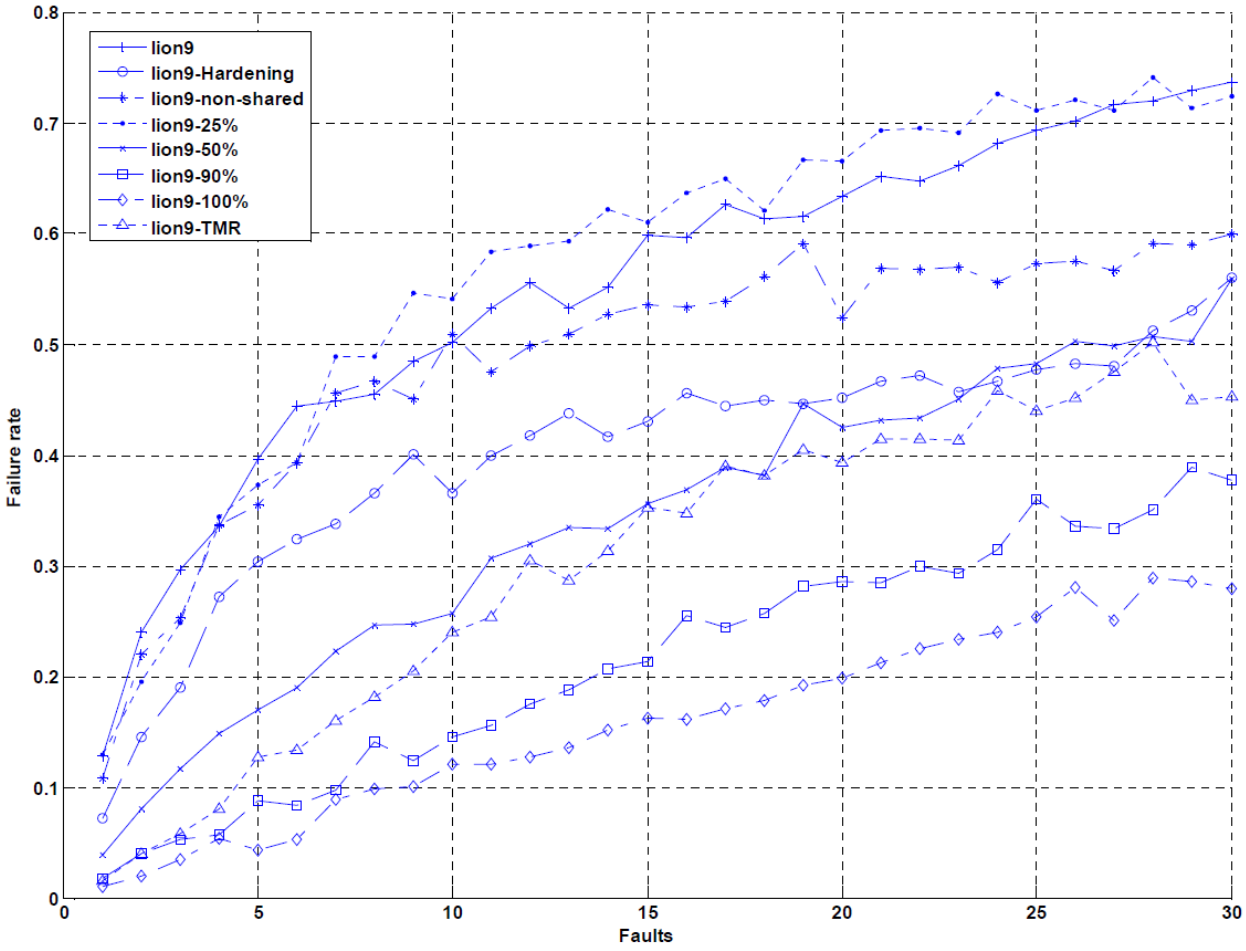


Figure 6.26: Failure rate vs Faults for lion9.

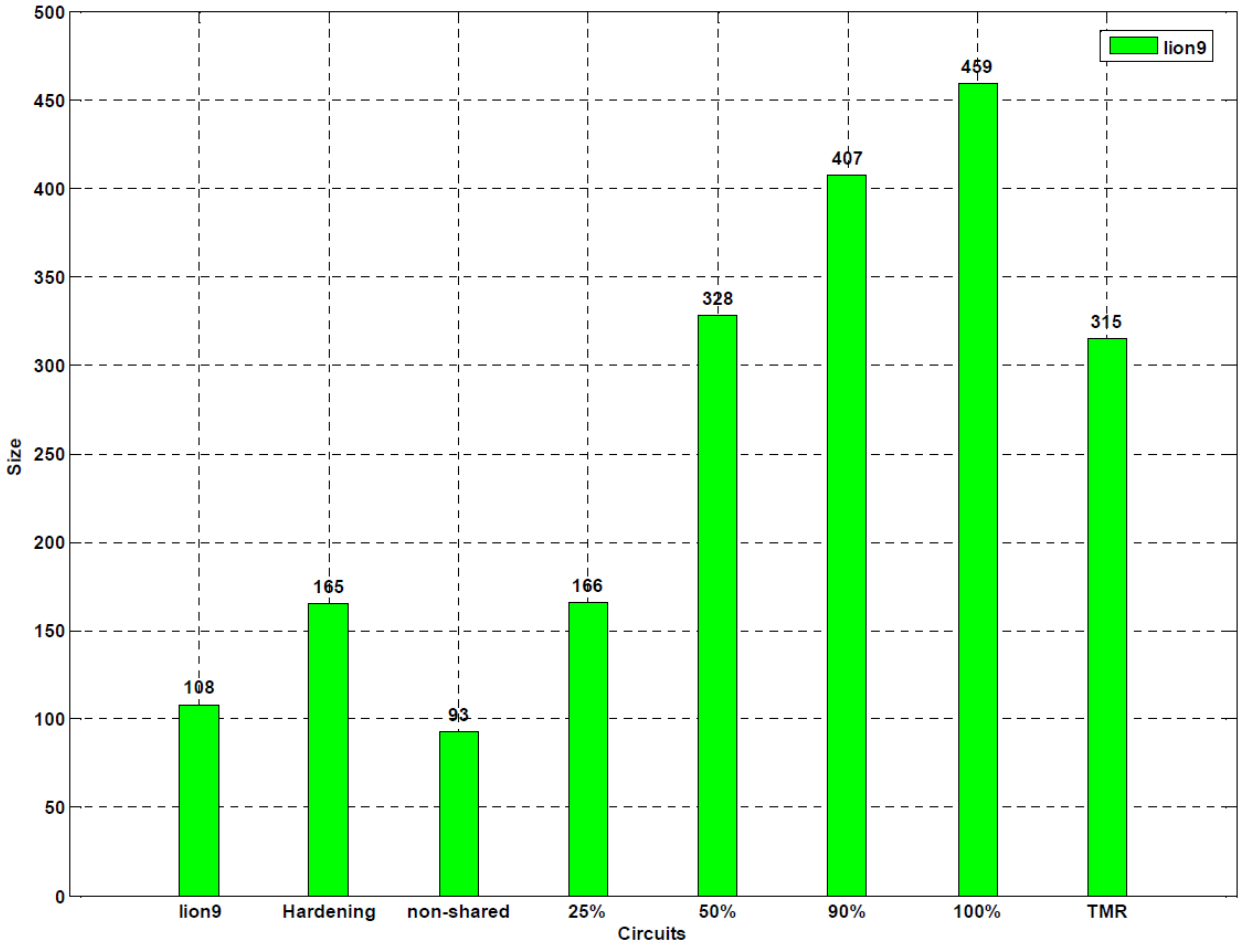


Figure 6.27: Size of lion9 experiments.

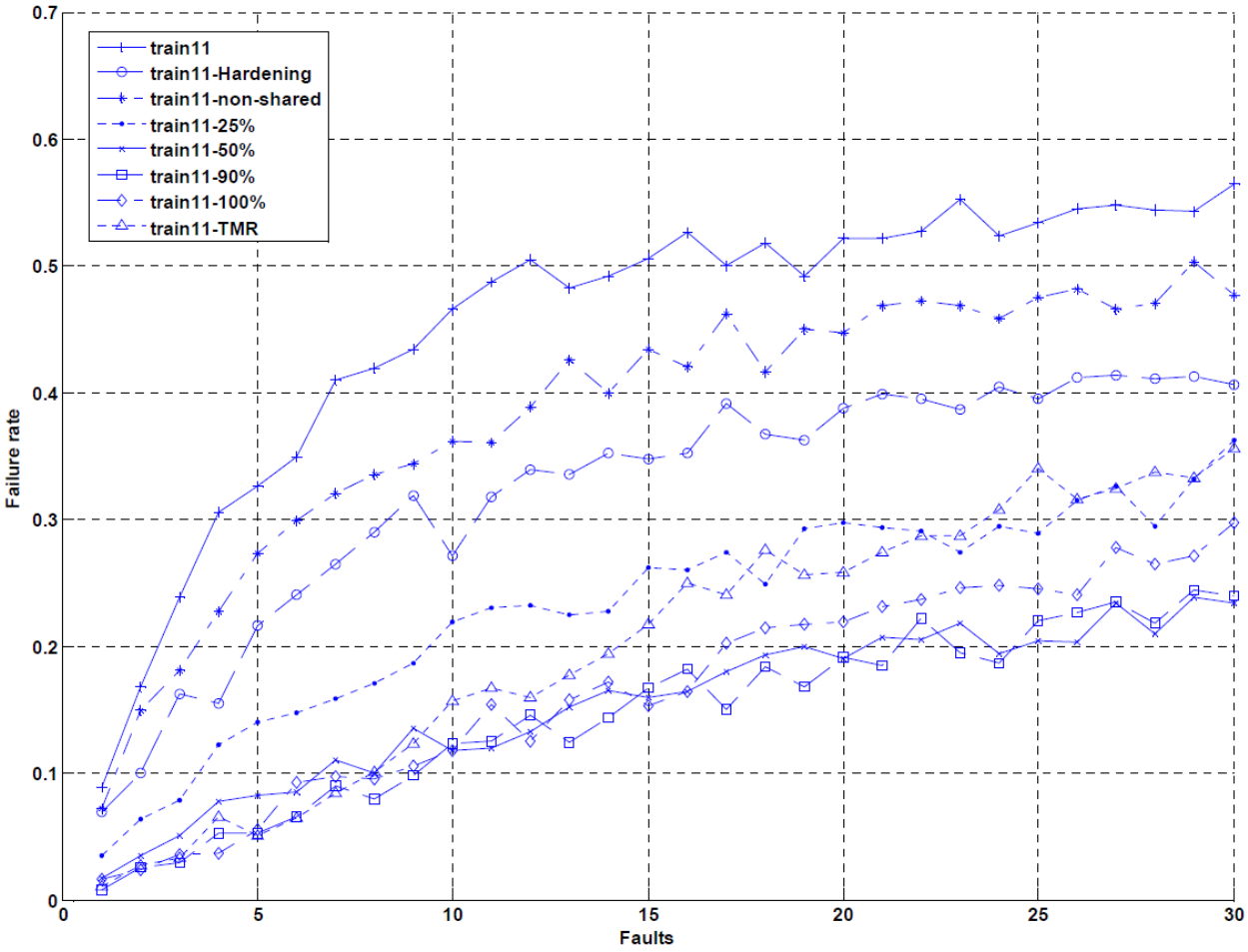


Figure 6.28: Failure rate vs Faults for train11.

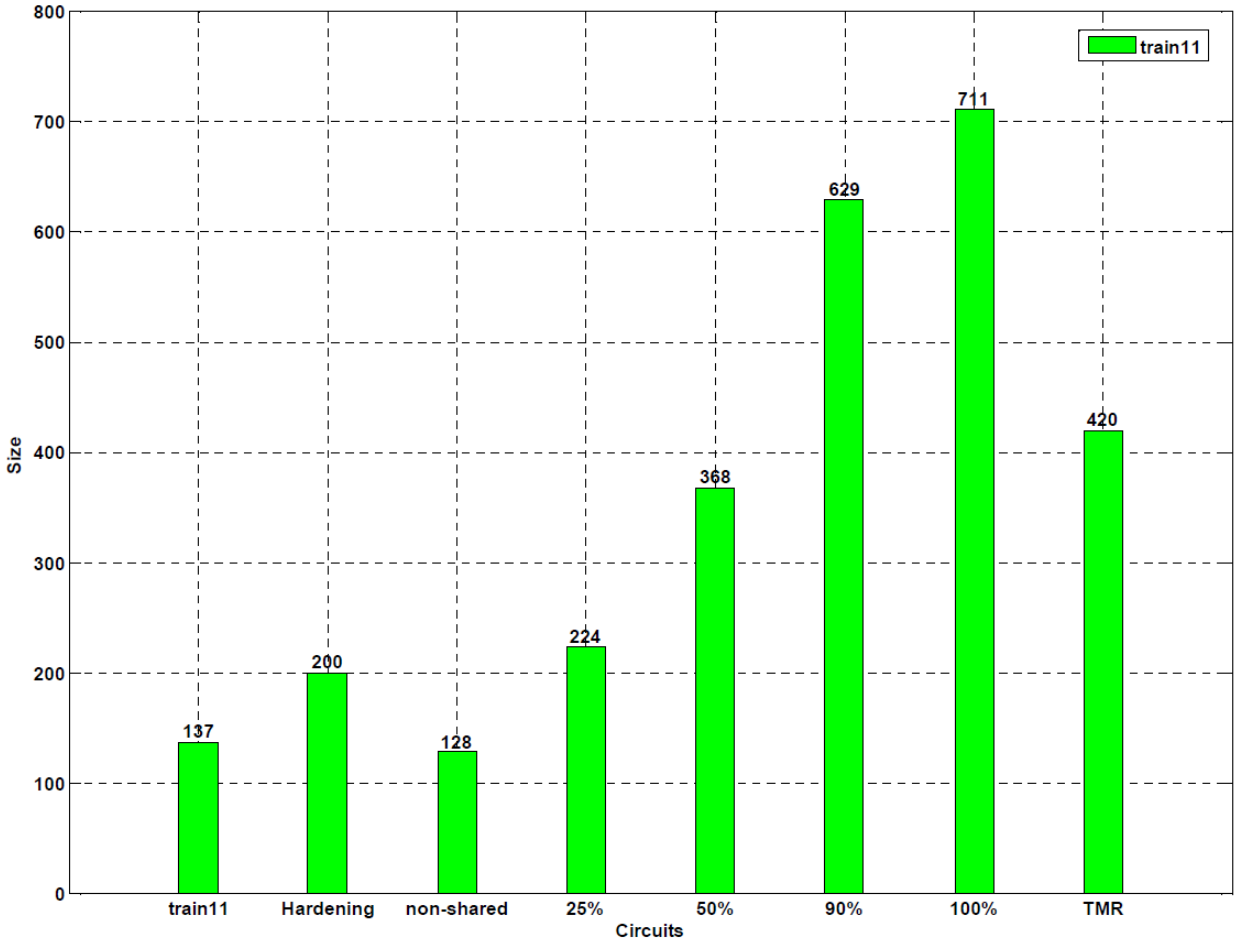


Figure 6.29: Size of train11 experiments.

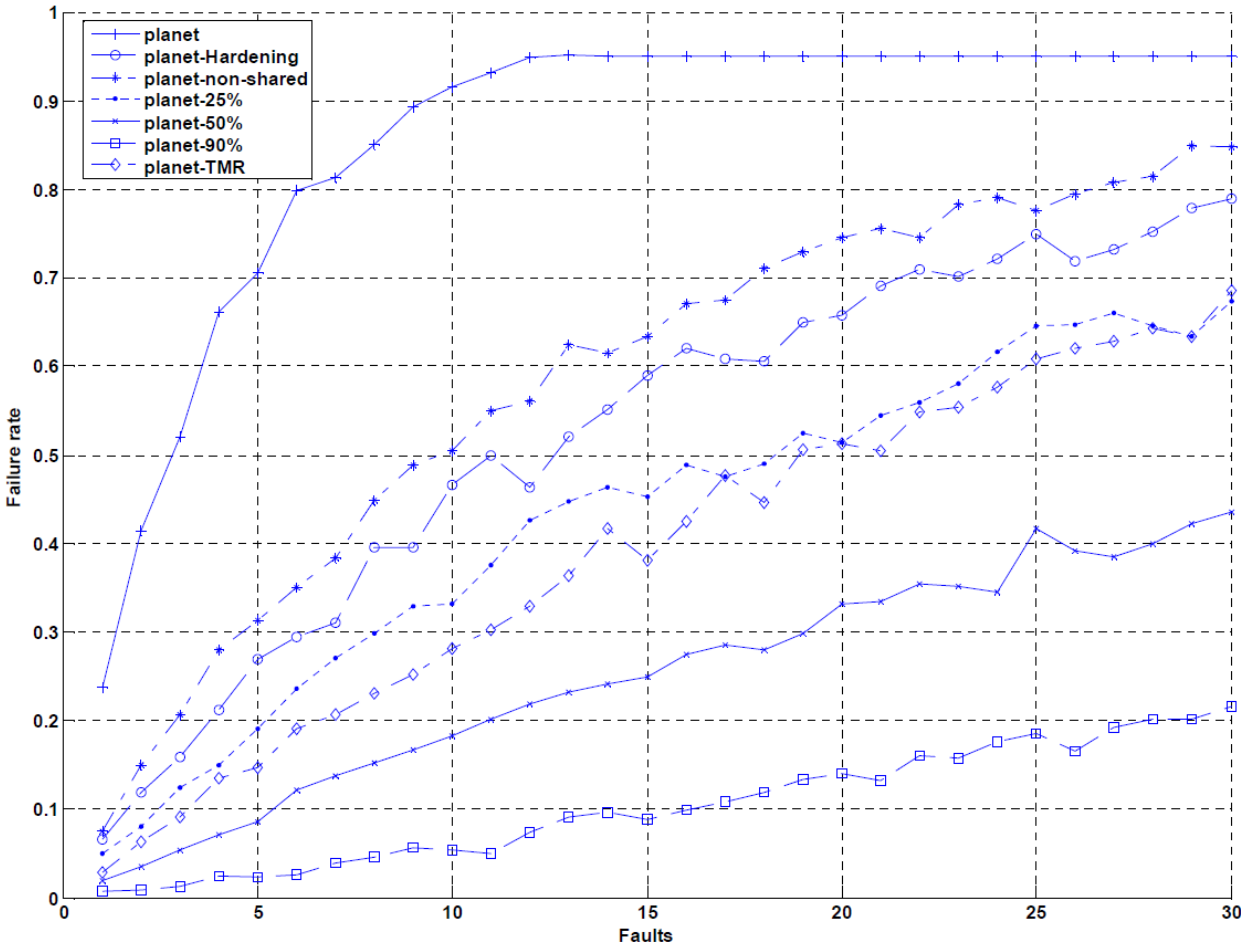


Figure 6.30: Failure rate vs Faults for planet.

covers almost 90% of state probabilities by protecting only and only 1 state with overall area cost that is 1.5 times compared to the original circuit.

The circuit tbk (See Figures 6.34 and 6.35) is a special circuit with the best logical masking among the circuits. The original circuit without sharing the logic among the memory elements gives around 0.025 failure rate when injecting 30 faults and 0.05 failure rate at injecting 60 faults. Because the logical masking of the circuit is good enough in the original circuit, TMRing fails in enhancing the reliability because they introduce voter circuits that present new hardly masked

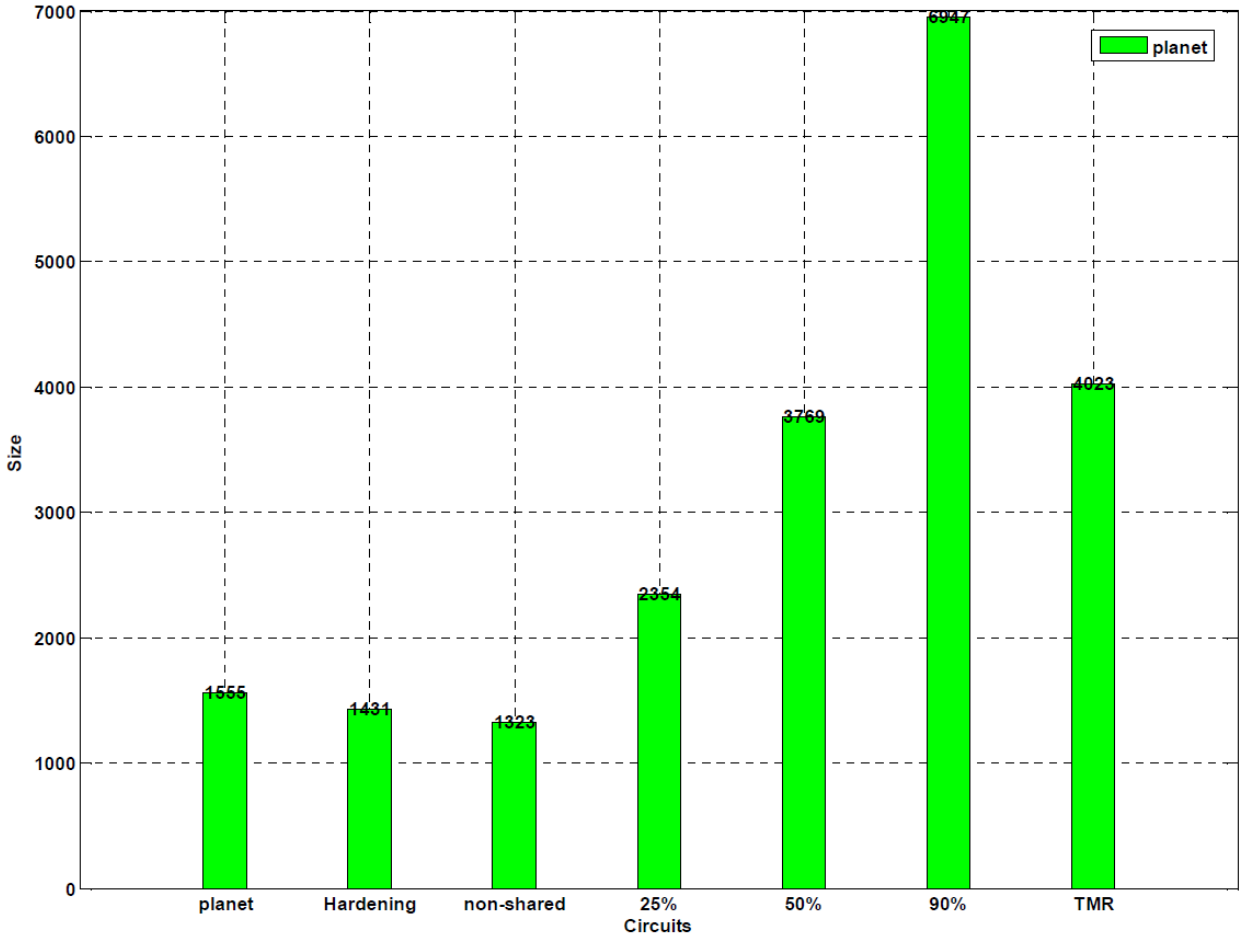


Figure 6.31: Size of planet experiments.

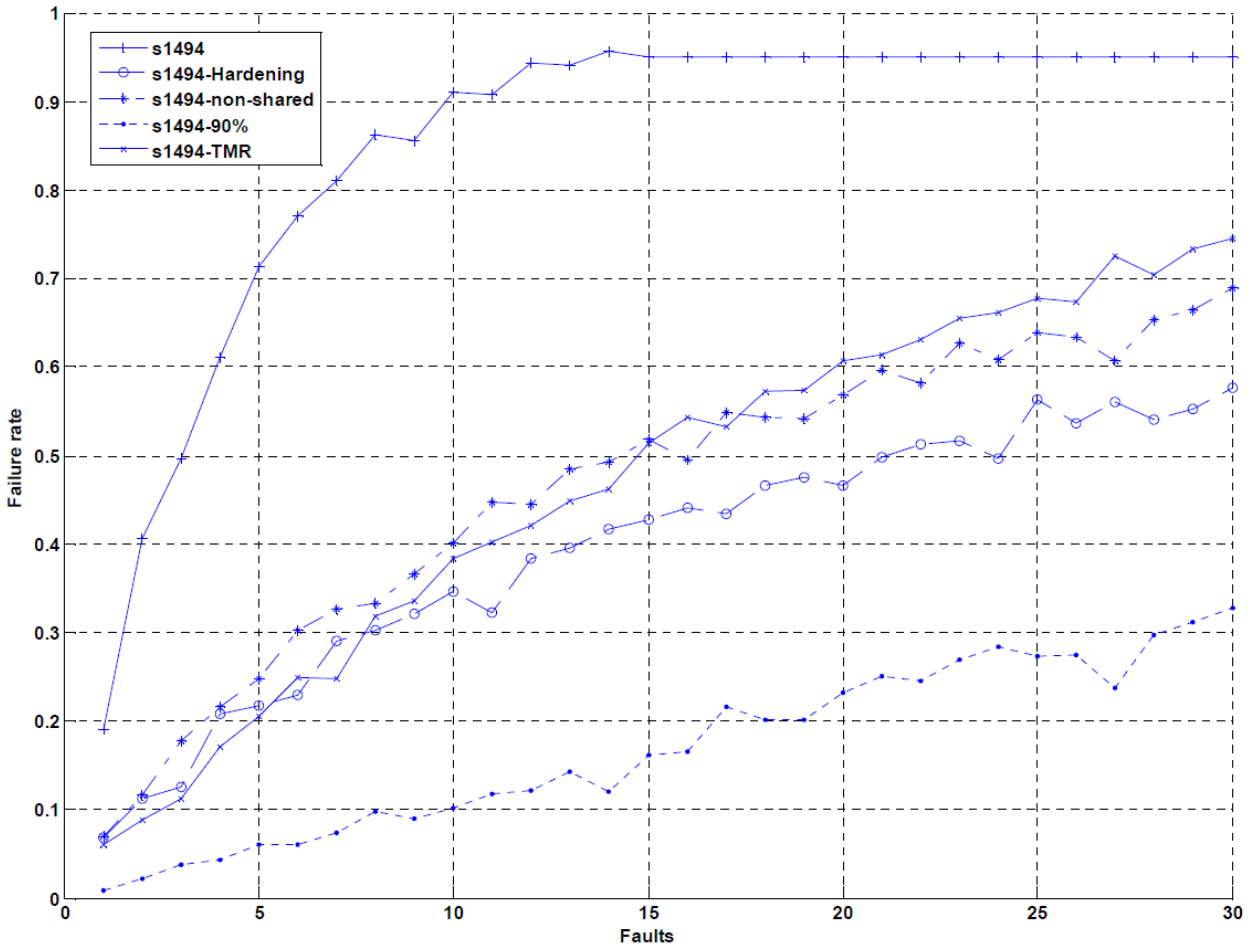


Figure 6.32: Failure rate vs Faults for s1494.

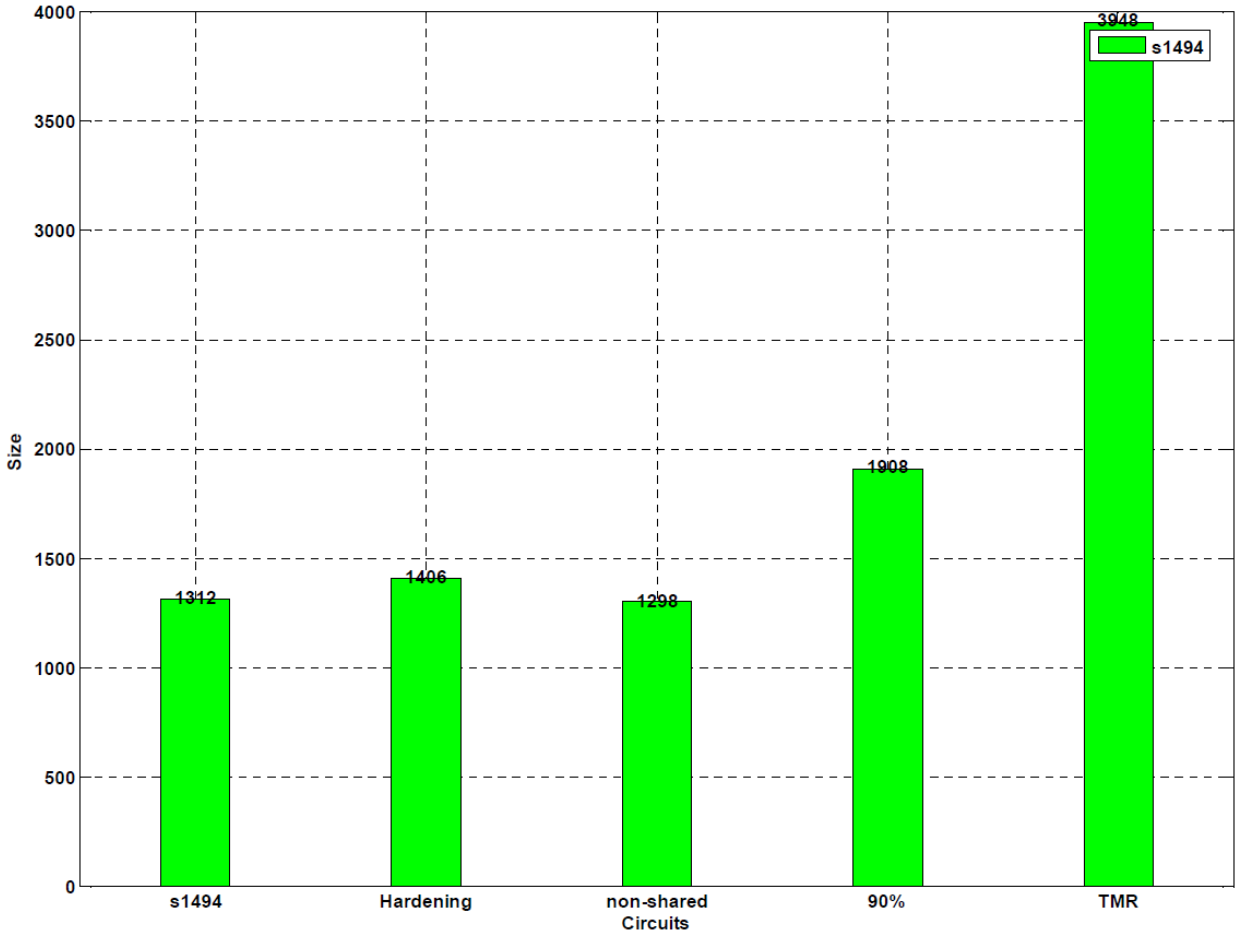


Figure 6.33: Size of s1494 experiments.

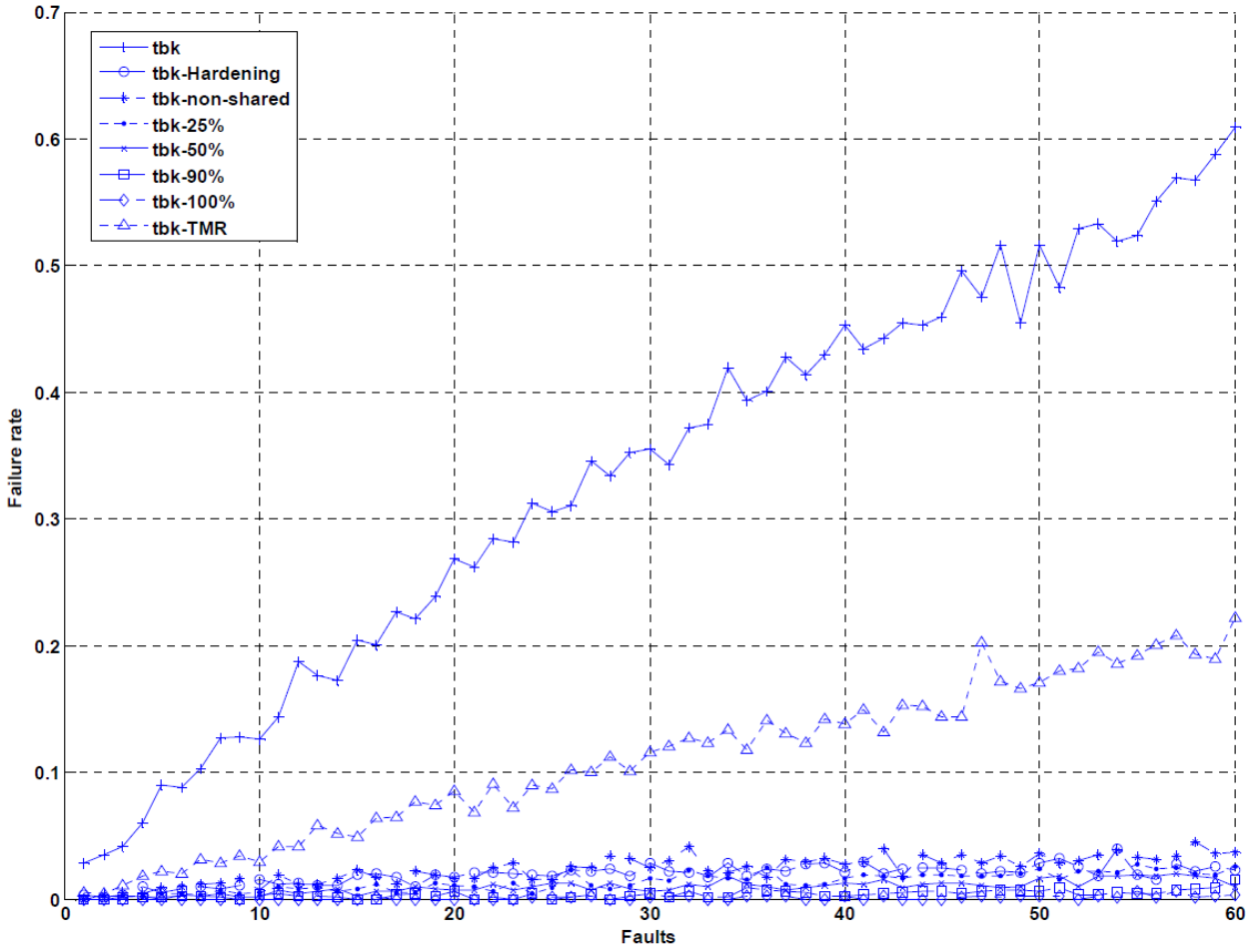


Figure 6.34: Failure rate vs Faults for tbk.

point of failure. The hardening methods gives a very close failure rate to the original circuit without sharing the logic. On the other hand, 25%, 50%, 90% as well as 100% all give better failure rate measurement.

6.3.3 Aggregated Results and Conclusions

In this section, we report the aggregated results from running FSM sequential benchmark circuits such as: bbara, bbsse, dk14, lion9, train11, cse, keyb, s1, planet, pma, s832, s1494, sand, styr and tbk. Results of failure rate from ISCAS89

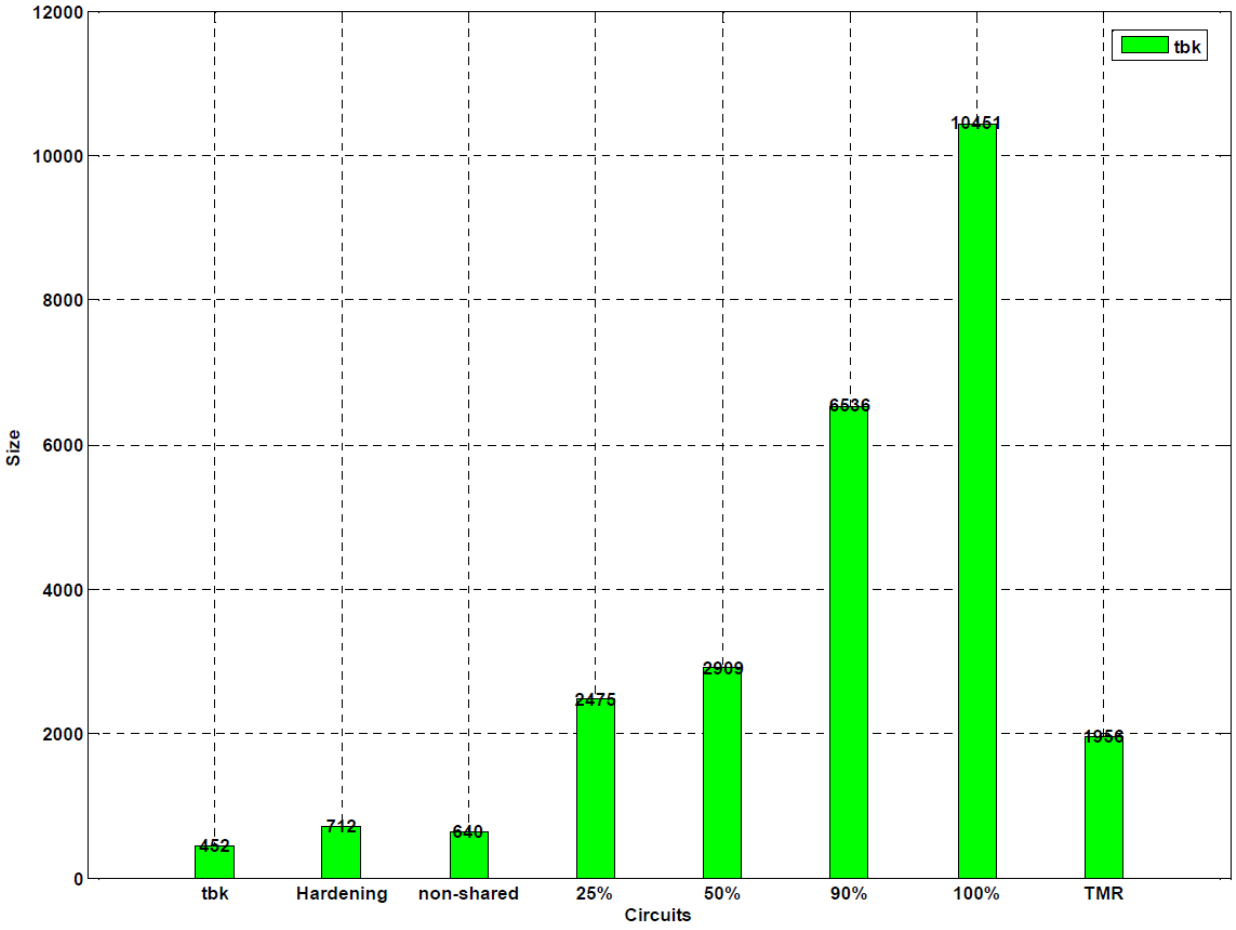


Figure 6.35: Size of tbk experiments.

sequential benchmark circuits for injecting 1, 5, 10, 20 and 30 faults are shown in Tables 6.6, 6.7, 6.8, 6.9 and 6.10. In those tables, the amount of reduction (-ve sign in the table) or increase from the original circuit in terms of failure rate is stated and averaged among all the circuits. For each circuit, the failure rate of the original circuit, the non-shared version of it, 25%, 50%, 90% and 100% probability coverage failure rate, hardening, and TMRing both the logic and the flip flops are reported.

From these tables, we can draw the following conclusions:

- By injecting 1 fault, the best failure rate reduction is 100% for 100% state probability coverage protection using Algorithm 1. This is followed by 94% failure rate reduction for 90% state probability coverage, 88% failure rate reduction for 50% state probability coverage, 86% failure rate reduction for TMRing both the logic and memory elements, 76% failure rate reduction for 25% state probability coverage, 66% failure rate reduction for hardening, then 60% failure rate reduction for non-shared logic original circuit.

- By injecting 5 faults, the best failure rate reduction is 96% for 100% state probability coverage protection using Algorithm 1. This is followed by 91% failure rate reduction for 90% state probability coverage, 82% failure rate reduction for 50% state probability coverage, 80% failure rate reduction for TMRing both the logic and memory elements, 71% failure rate reduction for 25% state probability coverage, 62% failure rate reduction for hardening, then 53% failure rate reduction for non-shared logic original circuit.

- By injecting 10 faults, the best failure rate reduction is 93% for 100% state probability coverage protection using Algorithm 1. This is followed by 86% failure rate reduction for 90% state probability coverage, 76% failure rate reduction for 50% state probability coverage, 70% failure rate reduction for TMRing both the logic and memory elements, 64% failure rate reduction for 25% state probability coverage, 55% failure rate reduction for hardening, then 45% failure rate reduction for non-shared logic original circuit.

- By injecting 20 faults, the best failure rate reduction is 87% for 100% state probability coverage protection using algorithm 1. lThis is followed by 76% failure rate reduction for 90% state probability coverage, 64% failure rate reduction for 50% state probability coverage, 55% failure rate reduction for TMRing both the logic and memory elements, 53% failure rate reduction for 25% state probability coverage, 46% failure rate reduction for hardening, then 36% failure rate reduction for non-shared logic original circuit.

- By injecting 30 faults, the best failure rate reduction is 81% for 100% state probability coverage protection using Algorithm 1. This is followed by 68% failure rate reduction for 90% state probability coverage, 55% failure rate reduction for 50% state probability coverage, 45% failure rate reduction for 25% state probability coverage, 43% failure rate reduction for TMRing both the logic and memory elements, 38% failure rate reduction for hardening, then 29% failure rate reduction for non-shared logic original circuit.

The area for each experiment in each circuit is shown in Table 6.11. The most

Table 6.6: Failure rate results for ISCAS89 sequential benchmark circuits - Injecting 1 fault.

Circuits	Original	Non-shared	State probability coverage				Comparison	
			25%	50%	90%	100%	Harden	TMR-FF-L
bbara	0.081	0.042	0.037	0.02	0.002	0	0.029	0.008
bbsse	0.177	0.069	0.035	0.016	0.003	0	0.058	0.011
cse	0.128	0.027	0.007	0.007	0.008	0	0.023	0.017
keyb	0.098	0.021	0.008	0.008	0.003	0	0.017	0.003
lion9	0.128	0.108	0.129	0.039	0.018	0	0.072	0.015
planet	0.238	0.075	0.05	0.019	0.007	0	0.066	0.029
pma	0.238	0.125	0.069	0.045	0.029	0	0.105	0.036
s1	0.14	0.057	0.035	0.016	0.005	0	0.065	0.015
s832	0.18	0.049	0.015	0.015	0.007	0	0.043	0.021
s1494	0.191	0.07	0.009	0.009	0.009	0	0.069	0.061
sand	0.167	0.05	0.025	0.006	0.004	0	0.043	0.023
styr	0.161	0.059	0.022	0.022	0.023	0	0.026	0.031
tbk	0.028	0	0	0	0	0	0.003	0.005
train11	0.089	0.072	0.035	0.017	0.008	0	0.069	0.011
dk14	0.189	0.099	0.053	0.022	0.01	0	0.076	0.039

Circuits	Original	Non-shared	State probability coverage				Comparison	
			25%	50%	90%	100%	Harden	TMR-FF-L
bbara	0%	-48%	-54%	-75%	-98%	-100%	-64%	-90%
bbsse	0%	-61%	-80%	-91%	-98%	-100%	-67%	-94%
cse	0%	-79%	-95%	-95%	-94%	-100%	-82%	-87%
keyb	0%	-79%	-92%	-92%	-97%	-100%	-83%	-97%
lion9	0%	-16%	1%	-70%	-86%	-100%	-44%	-88%
planet	0%	-68%	-79%	-92%	-97%	-100%	-72%	-88%
pma	0%	-47%	-71%	-81%	-88%	-100%	-56%	-85%
s1	0%	-59%	-75%	-89%	-96%	-100%	-54%	-89%
s832	0%	-73%	-92%	-92%	-96%	-100%	-76%	-88%
s1494	0%	-63%	-95%	-95%	-95%	-100%	-64%	-68%
sand	0%	-70%	-85%	-96%	-98%	-100%	-74%	-86%
styr	0%	-63%	-86%	-86%	-86%	-100%	-84%	-81%
tbk	0%	-100%	-100%	-100%	-100%	-100%	-89%	-82%
train11	0%	-19%	-61%	-81%	-91%	-100%	-22%	-88%
dk14	0%	-48%	-72%	-88%	-95%	-100%	-60%	-79%
Average	0%	-60%	-76%	-88%	-94%	-100%	-66%	-86%

Table 6.7: Failure rate results for ISCAS89 sequential benchmark circuits - Injecting 5 fault.

Circuits	Original	Non-shared	State probability coverage				Comparison	
			25%	50%	90%	100%	Harden	TMR-FF-L
bbara	0.304	0.177	0.122	0.098	0.014	0.002	0.137	0.059
bbsse	0.55	0.285	0.149	0.075	0.031	0.011	0.233	0.071
cse	0.451	0.127	0.051	0.051	0.025	0.014	0.081	0.089
keyb	0.317	0.11	0.031	0.031	0.025	0.002	0.087	0.035
lion9	0.396	0.355	0.373	0.17	0.088	0.043	0.304	0.127
planet	0.706	0.313	0.191	0.086	0.023	0.001	0.269	0.147
pma	0.709	0.424	0.312	0.178	0.113	0.067	0.371	0.155
s1	0.572	0.298	0.169	0.085	0.024	0.006	0.231	0.086
s832	0.592	0.195	0.095	0.095	0.043	0.002	0.14	0.092
s1494	0.714	0.248	0.06	0.06	0.06	0.002	0.217	0.206
sand	0.601	0.228	0.092	0.086	0.029	0.013	0.167	0.086
styr	0.559	0.21	0.118	0.118	0.076	0.042	0.197	0.118
tbk	0.09	0.009	0.004	0	0.001	0	0.005	0.022
train11	0.326	0.273	0.14	0.082	0.053	0.055	0.216	0.051
dk14	0.643	0.326	0.254	0.131	0.056	0.006	0.31	0.195

Circuits	Original	Non-shared	State probability coverage				Comparison	
			25%	50%	90%	100%	Harden	TMR-FF-L
bbara	0%	-42%	-60%	-68%	-95%	-99%	-55%	-81%
bbsse	0%	-48%	-73%	-86%	-94%	-98%	-58%	-87%
cse	0%	-72%	-89%	-89%	-94%	-97%	-82%	-80%
keyb	0%	-65%	-90%	-90%	-92%	-99%	-73%	-89%
lion9	0%	-10%	-6%	-57%	-78%	-89%	-23%	-68%
planet	0%	-56%	-73%	-88%	-97%	-100%	-62%	-79%
pma	0%	-40%	-56%	-75%	-84%	-91%	-48%	-78%
s1	0%	-48%	-70%	-85%	-96%	-99%	-60%	-85%
s832	0%	-67%	-84%	-84%	-93%	-100%	-76%	-84%
s1494	0%	-65%	-92%	-92%	-92%	-100%	-70%	-71%
sand	0%	-62%	-85%	-86%	-95%	-98%	-72%	-86%
styr	0%	-62%	-79%	-79%	-86%	-92%	-65%	-79%
tbk	0%	-90%	-96%	-100%	-99%	-100%	-94%	-76%
train11	0%	-16%	-57%	-75%	-84%	-83%	-34%	-84%
dk14	0%	-49%	-60%	-80%	-91%	-99%	-52%	-70%
Average	0%	-53%	-71%	-82%	-91%	-96%	-61%	-80%

Table 6.8: Failure rate results for ISCAS89 sequential benchmark circuits - Injecting 10 fault.

Circuits	Original	Non-shared	State probability coverage				Comparison	
			25%	50%	90%	100%	Harden	TMR-FF-L
bbara	0.431	0.283	0.18	0.177	0.051	0.015	0.189	0.108
bbsse	0.717	0.456	0.259	0.177	0.072	0.032	0.368	0.193
cse	0.644	0.212	0.092	0.092	0.074	0.028	0.171	0.187
keyb	0.454	0.21	0.086	0.086	0.061	0.005	0.154	0.083
lion9	0.502	0.509	0.541	0.257	0.145	0.121	0.365	0.24
planet	0.915	0.504	0.332	0.183	0.054	0.039	0.466	0.281
pma	0.89	0.635	0.544	0.353	0.221	0.121	0.593	0.336
s1	0.758	0.458	0.325	0.196	0.064	0.026	0.379	0.177
s832	0.816	0.324	0.154	0.154	0.103	0.015	0.232	0.163
s1494	0.91	0.401	0.102	0.102	0.102	0.088	0.346	0.384
sand	0.803	0.35	0.2	0.148	0.079	0.017	0.292	0.2
styr	0.762	0.367	0.214	0.214	0.146	0.077	0.33	0.229
tbk	0.126	0.007	0.005	0.001	0.003	0	0.015	0.029
train11	0.466	0.362	0.219	0.118	0.123	0.118	0.271	0.157
dk14	0.832	0.548	0.438	0.243	0.135	0.031	0.495	0.367

Circuits	Original	Non-shared	State probability coverage				Comparison	
			25%	50%	90%	100%	Harden	TMR-FF-L
bbara	0%	-34%	-58%	-59%	-88%	-97%	-56%	-75%
bbsse	0%	-36%	-64%	-75%	-90%	-96%	-49%	-73%
cse	0%	-67%	-86%	-86%	-89%	-96%	-73%	-71%
keyb	0%	-54%	-81%	-81%	-87%	-99%	-66%	-82%
lion9	0%	1%	8%	-49%	-71%	-76%	-27%	-52%
planet	0%	-45%	-64%	-80%	-94%	-96%	-49%	-69%
pma	0%	-29%	-39%	-60%	-75%	-86%	-33%	-62%
s1	0%	-40%	-57%	-74%	-92%	-97%	-50%	-77%
s832	0%	-60%	-81%	-81%	-87%	-98%	-72%	-80%
s1494	0%	-56%	-89%	-89%	-89%	-90%	-62%	-58%
sand	0%	-56%	-75%	-82%	-90%	-98%	-64%	-75%
styr	0%	-52%	-72%	-72%	-81%	-90%	-57%	-70%
tbk	0%	-94%	-96%	-99%	-98%	-100%	-88%	-77%
train11	0%	-22%	-53%	-75%	-74%	-75%	-42%	-66%
dk14	0%	-34%	-47%	-71%	-84%	-96%	-41%	-56%
Average	0%	-45%	-64%	-75%	-86%	-93%	-55%	-70%

Table 6.9: Failure rate results for ISCAS89 sequential benchmark circuits - Injecting 20 fault.

Circuits	Original	Non-shared	State probability coverage				Comparison	
			25%	50%	90%	100%	Harden	TMR-FF-L
bbara	0.516	0.359	0.277	0.236	0.104	0.058	0.316	0.187
bbsse	0.88	0.602	0.44	0.311	0.196	0.052	0.484	0.362
cse	0.818	0.346	0.175	0.175	0.157	0.051	0.24	0.319
keyb	0.657	0.354	0.172	0.172	0.11	0.024	0.301	0.155
lion9	0.633	0.524	0.665	0.425	0.286	0.198	0.452	0.393
planet	0.95	0.745	0.514	0.332	0.14	0.07	0.657	0.513
pma	0.95	0.845	0.757	0.599	0.41	0.252	0.806	0.578
s1	0.928	0.668	0.496	0.294	0.149	0.068	0.59	0.34
s832	0.949	0.501	0.273	0.273	0.216	0.028	0.351	0.37
s1494	0.95	0.568	0.232	0.232	0.232	0.102	0.466	0.607
sand	0.958	0.545	0.365	0.274	0.146	0.074	0.497	0.335
styr	0.893	0.609	0.393	0.393	0.244	0.136	0.516	0.385
tbk	0.269	0.016	0.011	0.007	0.006	0	0.017	0.085
train11	0.522	0.447	0.297	0.19	0.191	0.219	0.388	0.258
dk14	0.95	0.701	0.626	0.438	0.275	0.127	0.58	0.599

Circuits	Original	Non-shared	State probability coverage				Comparison	
			25%	50%	90%	100%	Harden	TMR-FF-L
bbara	0%	-30%	-46%	-54%	-80%	-89%	-39%	-64%
bbsse	0%	-32%	-50%	-65%	-78%	-94%	-45%	-59%
cse	0%	-58%	-79%	-79%	-81%	-94%	-71%	-61%
keyb	0%	-46%	-74%	-74%	-83%	-96%	-54%	-76%
lion9	0%	-17%	5%	-33%	-55%	-69%	-29%	-38%
planet	0%	-22%	-46%	-65%	-85%	-93%	-31%	-46%
pma	0%	-11%	-20%	-37%	-57%	-73%	-15%	-39%
s1	0%	-28%	-47%	-68%	-84%	-93%	-36%	-63%
s832	0%	-47%	-71%	-71%	-77%	-97%	-63%	-61%
s1494	0%	-40%	-76%	-76%	-76%	-89%	-51%	-36%
sand	0%	-43%	-62%	-71%	-85%	-92%	-48%	-65%
styr	0%	-32%	-56%	-56%	-73%	-85%	-42%	-57%
tbk	0%	-94%	-96%	-97%	-98%	-100%	-94%	-68%
train11	0%	-14%	-43%	-64%	-63%	-58%	-26%	-51%
dk14	0%	-26%	-34%	-54%	-71%	-87%	-39%	-37%
Average	0%	-36%	-53%	-64%	-76%	-87%	-45%	-55%

Table 6.10: Failure rate results for ISCAS89 sequential benchmark circuits - Injecting 30 fault.

Circuits	Original	Non-shared	State probability coverage				Comparison	
			25%	50%	90%	100%	Harden	TMR-FF-L
bbara	0.559	0.464	0.323	0.269	0.182	0.103	0.358	0.322
bbsse	0.94	0.681	0.538	0.445	0.318	0.119	0.6	0.497
cse	0.896	0.449	0.251	0.251	0.221	0.075	0.303	0.442
keyb	0.751	0.486	0.26	0.26	0.154	0.047	0.38	0.2
lion9	0.737	0.599	0.724	0.559	0.377	0.279	0.56	0.453
planet	0.95	0.848	0.674	0.435	0.216	0.104	0.789	0.686
pma	0.95	0.924	0.841	0.72	0.545	0.354	0.884	0.743
s1	0.95	0.747	0.601	0.431	0.214	0.132	0.71	0.487
s832	0.95	0.609	0.39	0.39	0.293	0.08	0.471	0.469
s1494	0.95	0.689	0.328	0.328	0.328	0.211	0.577	0.745
sand	0.95	0.662	0.489	0.362	0.197	0.119	0.584	0.459
styr	0.947	0.706	0.518	0.518	0.381	0.212	0.636	0.515
tbk	0.355	0.025	0.016	0.006	0.004	0.001	0.028	0.116
train11	0.565	0.477	0.363	0.234	0.24	0.297	0.406	0.356
dk14	0.95	0.751	0.7	0.553	0.35	0.21	0.681	0.768

Circuits	Original	Non-shared	State probability coverage				Comparison	
			25%	50%	90%	100%	Harden	TMR-FF-L
bbara	0%	-17%	-42%	-52%	-67%	-82%	-36%	-42%
bbsse	0%	-28%	-43%	-53%	-66%	-87%	-36%	-47%
cse	0%	-50%	-72%	-72%	-75%	-92%	-66%	-51%
keyb	0%	-35%	-65%	-65%	-79%	-94%	-49%	-73%
lion9	0%	-19%	-2%	-24%	-49%	-62%	-24%	-39%
planet	0%	-11%	-29%	-54%	-77%	-89%	-17%	-28%
pma	0%	-3%	-11%	-24%	-43%	-63%	-7%	-22%
s1	0%	-21%	-37%	-55%	-77%	-86%	-25%	-49%
s832	0%	-36%	-59%	-59%	-69%	-92%	-50%	-51%
s1494	0%	-27%	-65%	-65%	-65%	-78%	-39%	-22%
sand	0%	-30%	-49%	-62%	-79%	-87%	-39%	-52%
styr	0%	-25%	-45%	-45%	-60%	-78%	-33%	-46%
tbk	0%	-93%	-95%	-98%	-99%	-100%	-92%	-67%
train11	0%	-16%	-36%	-59%	-58%	-47%	-28%	-37%
dk14	0%	-21%	-26%	-42%	-63%	-78%	-28%	-19%
Average	0%	-29%	-45%	-55%	-69%	-81%	-38%	-43%

area overhead on average is 1185% for 100% state probability coverage protection using Algorithm 1. This is followed by 315% for 90% state probability coverage, 219% for TMRing both the logic and memory elements, 143% for 50% state probability coverage, 83% for 25% state probability coverage, 24% for hardening, then 3% area overhead for non-shared logic original circuit.

It is concluded that protecting states yielding 90% state probability coverage increase the area of the original circuit for about 4 times. Protecting states yielding 50% state probability coverage almost increases the area of the original circuit 2.5 times. Protecting states yielding 25% state probability coverage almost increases the area of the original circuit 1.8 time. On the other hand, TMRing both combinational logic and memory elements increases the area of the original circuit 3 times which is between the area overhead cost of protecting states yielding 90% state probability coverage and 50%.

In Table 6.12, the number of protected states in each state probability coverage is summarized and averaged among the circuits. The following points can be concluded:

- To cover 25% state probability of a FSM, 2 states on average should be protected. This is 9% of states in the FSM.
- To cover 50% state probability of a FSM, 4 states on average should be protected. This is 21% of states in the FSM.
- To cover 90% state probability of a FSM, 9 states on average should be protected. This is 40% of states in the FSM.

Table 6.11: Area overhead for ISCAS89 sequential benchmark circuits.

Circuits	Original	Non-shared	State probability coverage				Comparison	
			25%	50%	90%	100%	Harden	TMR-FF-L
bbara	179	174	282	401	695	1400	246	558
bbsse	352	363	455	577	637	1846	435	1125
cse	483	525	740	740	862	3263	597	1611
keyb	565	715	1033	1033	1128	5081	805	2190
lion9	108	93	166	328	407	459	165	315
planet	1555	1323	2354	3769	6947	120376	1431	4023
pma	489	483	717	978	1618	3402	573	1494
s1	811	761	1317	2287	6037	6041	851	2328
s832	598	703	937	937	1236	5889	793	2154
s1494	1312	1298	1908	1908	1908	11738	1406	3948
sand	1121	1081	2328	3300	5382	9323	1171	3288
styr	1205	1228	2043	2043	4193	9239	1318	3729
tbk	452	640	2475	2909	6536	10451	712	1956
train11	137	128	224	368	629	711	200	420
dk14	280	262	341	598	822	1341	316	813

Circuits	Original	Non-shared	State probability coverage				Comparison	
			25%	50%	90%	100%	Harden	TMR-FF-L
bbara	0%	-3%	58%	124%	288%	682%	37%	212%
bbsse	0%	3%	29%	64%	81%	424%	24%	220%
cse	0%	9%	53%	53%	78%	576%	24%	234%
keyb	0%	27%	83%	83%	100%	799%	42%	288%
lion9	0%	-14%	54%	204%	277%	325%	53%	192%
planet	0%	-15%	51%	142%	347%	7641%	-8%	159%
pma	0%	-1%	47%	100%	231%	596%	17%	206%
s1	0%	-6%	62%	182%	644%	645%	5%	187%
s832	0%	18%	57%	57%	107%	885%	33%	260%
s1494	0%	-1%	45%	45%	45%	795%	7%	201%
sand	0%	-4%	108%	194%	380%	732%	4%	193%
styr	0%	2%	70%	70%	248%	667%	9%	209%
tbk	0%	42%	448%	544%	1346%	2212%	58%	333%
train11	0%	-7%	64%	169%	359%	419%	46%	207%
dk14	0%	-6%	22%	114%	194%	379%	13%	190%
Average	0%	3%	83%	143%	315%	1185%	24%	219%

Table 6.12: Number of protected states for ISCAS89 sequential benchmark circuits.

Circuits	State probability coverage				State percentage coverage			
	25%	50%	90%	100%	25%	50%	90%	100%
bbara	1	3	5	10	10%	30%	50%	100%
bbsse	1	2	3	13	8%	15%	23%	100%
cse	1	1	2	16	6%	6%	13%	100%
keyb	1	1	2	19	5%	5%	11%	100%
lion9	2	5	8	9	22%	56%	89%	100%
planet	5	12	30	48	10%	25%	63%	100%
pma	2	4	9	24	8%	17%	38%	100%
s1	2	5	12	20	10%	25%	60%	100%
s832	1	1	2	25	4%	4%	8%	100%
s1494	1	1	1	48	2%	2%	2%	100%
sand	3	8	19	32	9%	25%	59%	100%
styr	1	1	1	30	3%	3%	3%	100%
tbk	1	2	14	32	3%	6%	44%	100%
train11	2	6	9	11	18%	55%	82%	100%
dk14	1	3	4	7	14%	43%	57%	100%
Average	2	4	9	23	9%	21%	40%	100%

- To cover 100% state probability of a FSM, 23 states on average should be protected. This is 100% of states in the FSM.

The overall averages derived from the previously mentioned tables are shown in Table 6.13. We can say that on average, the best failure rate reduction is 91.4% for 100% state probability coverage protection using Algorithm 1. This is followed by 83% failure rate reduction for 90% state probability coverage, 73% failure rate reduction for 50% state probability coverage, 66.8% failure rate reduction for TMRing both the logic and memory elements, 61.8% failure rate reduction for 25% state probability coverage, 53.4% failure rate reduction for hardening, 44.6% failure rate reduction for non-shared logic original circuit.

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

- Protecting all the states guarantees 100% protection against single fault

Table 6.13: Overall averaged failure rate results for ISCAS89 sequential benchmark circuits.

Injected Faults	Original	Non-shared	State probability coverage				Comparison	
			25%	50%	90%	100%	Harden	TMR-FF-L
1	0	-60%	-76%	-88%	-94%	-100%	-66%	-86%
5	0	-53%	-71%	-82%	-91%	-96%	-62%	-80%
10	0	-45%	-64%	-76%	-86%	-93%	-55%	-70%
20	0	-36%	-53%	-64%	-76%	-87%	-46%	-55%
30	0	-29%	-45%	-55%	-68%	-81%	-38%	-43%
Average	0	-44.6%	-61.8%	-73.0%	-83.0%	-91.4%	-53.4%	-66.8%

errors.

- The best failure rate reduction is 91.4% when 100% state probability protection is chosen. However, the area overhead is about 13 times the original circuit and all the states must be protected by redundant states. We can avoid this by protecting states yielding 90% (50% or 25%) state probability coverage with area cost of about 4 times the original circuit(2.5 time for 50% or just adding 1.8 the original area for 25%). The number of states that should be protected in 90% probability state coverage is 40% of the states in the FSM (21% for 50% state probability coverage or 9% for 25%).

- Even thought we assume the best case scenario in hardening, protecting the memory elements alone does not give enough fault tolerance. Hardening enhances failure rate by 53.4% with area overhead of 1.24 times compared to the original sequential circuits with sharing combinational logic between the memory elements. However, by not sharing the combinational logic between the memory element without protecting them, the failure rate reduction is 44.6% (10% difference from hardening) with only area overhead of 1.03

compared to the shared original circuit.

- Protecting states yielding 90% state probability coverage is recommended since it reduces the failure rate dramatically by 83% when compared to original circuit which is better than TMRing both logic and memory elements. However, the area overhead when protecting states that yield to 90% state probability coverage is 4 times the original circuit size while TMRing area overhead is 3 times.

- If less area overhead is a requirement in the design of sequential circuit, protecting states yielding 50% state probability is recommended since it reduces the failure rate by 73% while the area overhead is below 2.5 times the original circuit.

6.4 Algorithm 2 Results

In this section, we will apply Algorithm 2 to two of the sequential benchmark circuits. The resulting findings for the circuits dk14 and bbara are discussed in the following section.

6.4.1 Comparison Between Different State Encodings

For each of the circuits (i.e. dk14 and bbara), four different random state encodings are considered and are compared to the encoding resulting from applying nova for state assignment in terms of failure rate and area. The objective is

Table 6.14: Codes used in Algorithm 2 for dk14.

states	dk14		codes before protection					codes after protection				
	prob.	accum.	code 1	code 2	code 3	code 4	nova	code 1	code 2	code 3	code 4	nova
S3	2.43E-01	2.43E-01	010	101	001	101	001	101010	101101	001001	011101	011001
S1	1.88E-01	4.31E-01	000	111	111	111	101	000000	000111	000111	000111	000101
S2	1.88E-01	6.19E-01	001	110	000	001	111	110001	110110	010000	100001	110111
S5	1.87E-01	8.06E-01	100	011	101	010	011	011100	011011	111101	001010	101011
S4	1.25E-01	9.31E-01	011	100	110	100	100	000011	000100	001110	000100	001100
S6	5.38E-02	9.85E-01	101	010	010	110	010	000101	000010	000010	010110	000010
S7	1.57E-02	1.00E+00	110	001	100	000	000	000110	000001	000100	000000	000000

to show the impact of starting with an already area optimized sequential circuit using nova state assignment on the resulting area after adding fault tolerance. For each state encoding, protecting states which yield to 90% state probability coverage is chosen.

For circuit dk14, the failure rate for the different types of encodings (see Table 6.14) are shown in Figure 6.36. Their failure rate despite their encodings are very close to each other. However, it is clearly depicted in Figure 6.37 that nova got the lowest area among the different encodings after applying protection using Algorithm 2 because it is originally the best area optimization that gives the minimum possible area overhead in the original circuit.

Likewise, bbara states are given different encodings (see Table 6.15) and their corresponding failure rates are shown in Figure 6.36. Again the area of nova is the minimum as it is shown in Figure 6.38.

We conclude that regardless of the encoding of states, the failure rate of protecting the same states is similar for different state encodings. By starting from

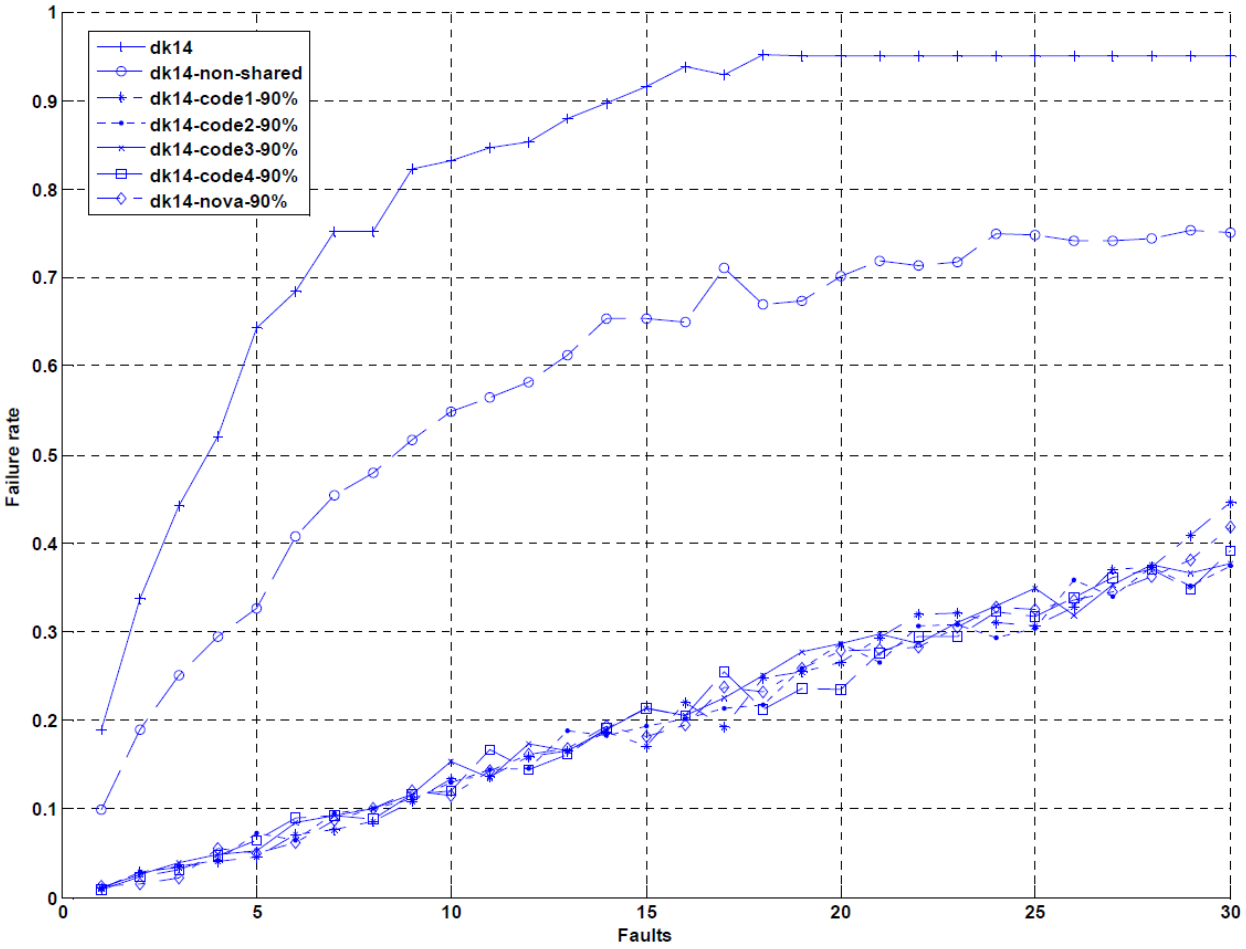


Figure 6.36: Failure rate vs Faults for dk14.

Table 6.15: Codes used in Algorithm 2 for bbara.

states	bbara		codes before protection					codes after protection				
	prob.	accum.	code 1	code 2	code 3	code 4	nova	code 1	code 2	code 3	code 4	nova
S1	2.67E-01	2.67E-01	0000	1001	1111	0110	0000	10100000	0001001	10101111	0000110	1010000
S4	1.97E-01	4.64E-01	0001	1000	1110	0111	0001	01000001	0111000	01001110	0110111	0110001
S0	1.55E-01	6.19E-01	0010	0111	1101	1000	0100	00000010	0000111	00001101	0001000	0000100
S2	1.33E-01	7.52E-01	0011	0110	1100	1001	0010	00110011	1100110	00111100	1101001	0100010
S3	1.33E-01	8.85E-01	0100	0101	1011	1010	0011	00010100	0110101	00011011	0111010	1000011
S5	4.92E-02	9.34E-01	0101	0100	1010	1011	1101	00000101	0000100	00001010	0001011	0001101
S7	3.74E-02	9.72E-01	0110	0011	1001	1100	0111	00100110	0010011	00101001	0011100	0000111
S6	1.64E-02	9.88E-01	0111	0010	1000	1101	1100	00000111	0000010	00001000	0001101	0011100
S8	9.36E-03	9.97E-01	1000	0001	0111	1110	0110	00001000	0010001	00000111	0011110	0010110
S9	2.34E-03	1.00E+00	1001	0000	0110	1111	0101	00001001	0000000	00000110	0001111	0010101
								8 bit	7 bit	8 bit	7 bit	7 bit

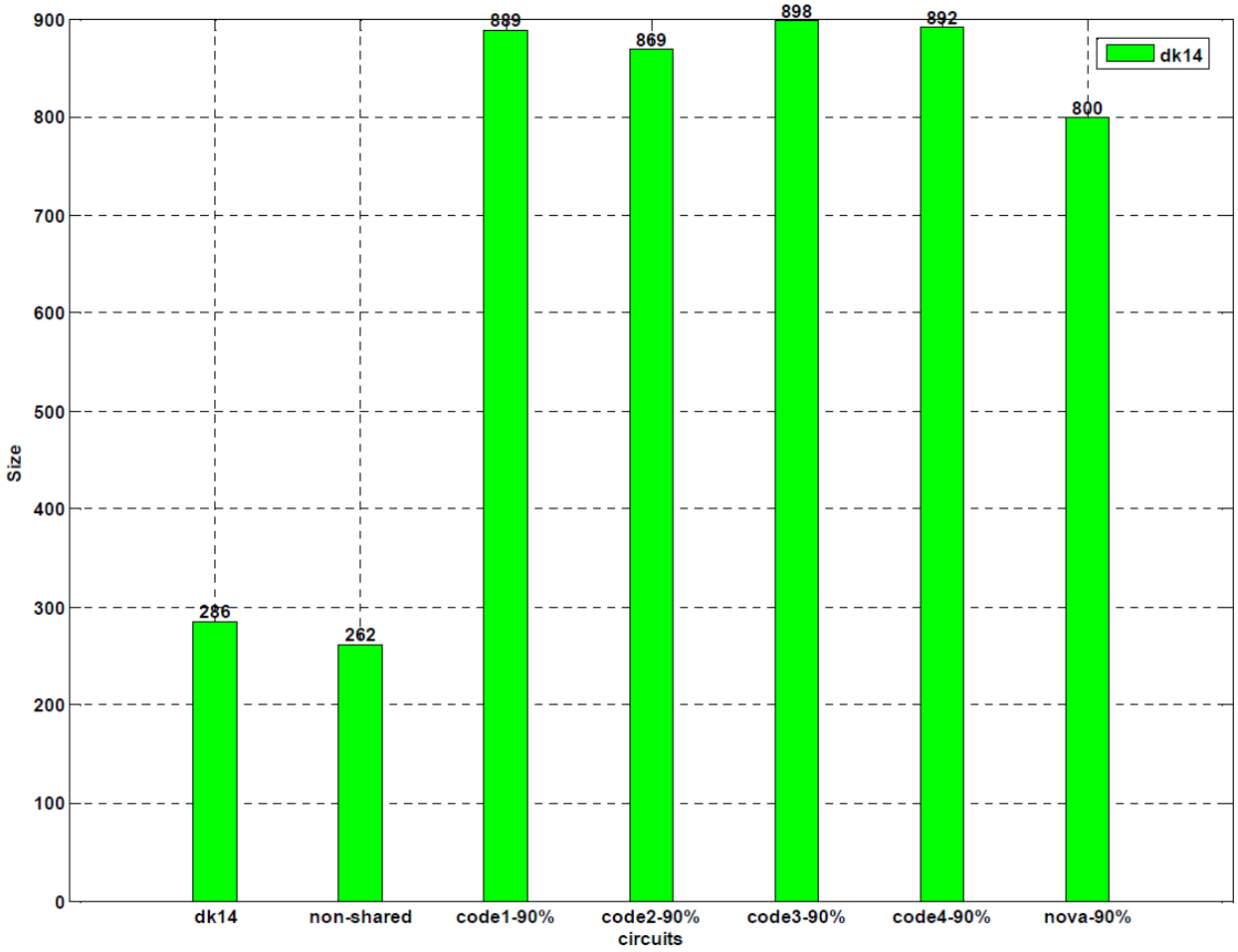


Figure 6.37: Size of dk14 experiments.

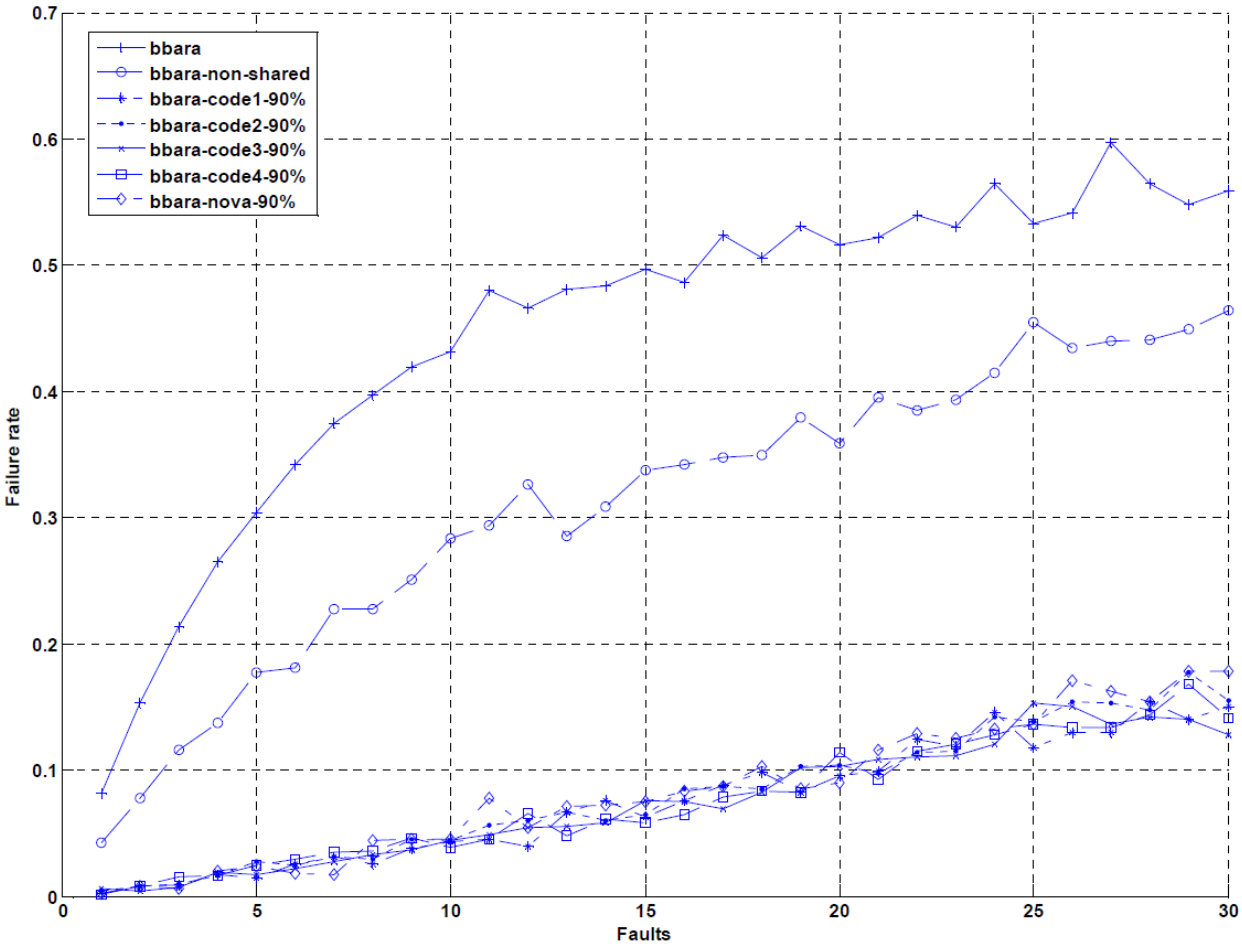


Figure 6.38: Failure rate vs Faults for bbara.

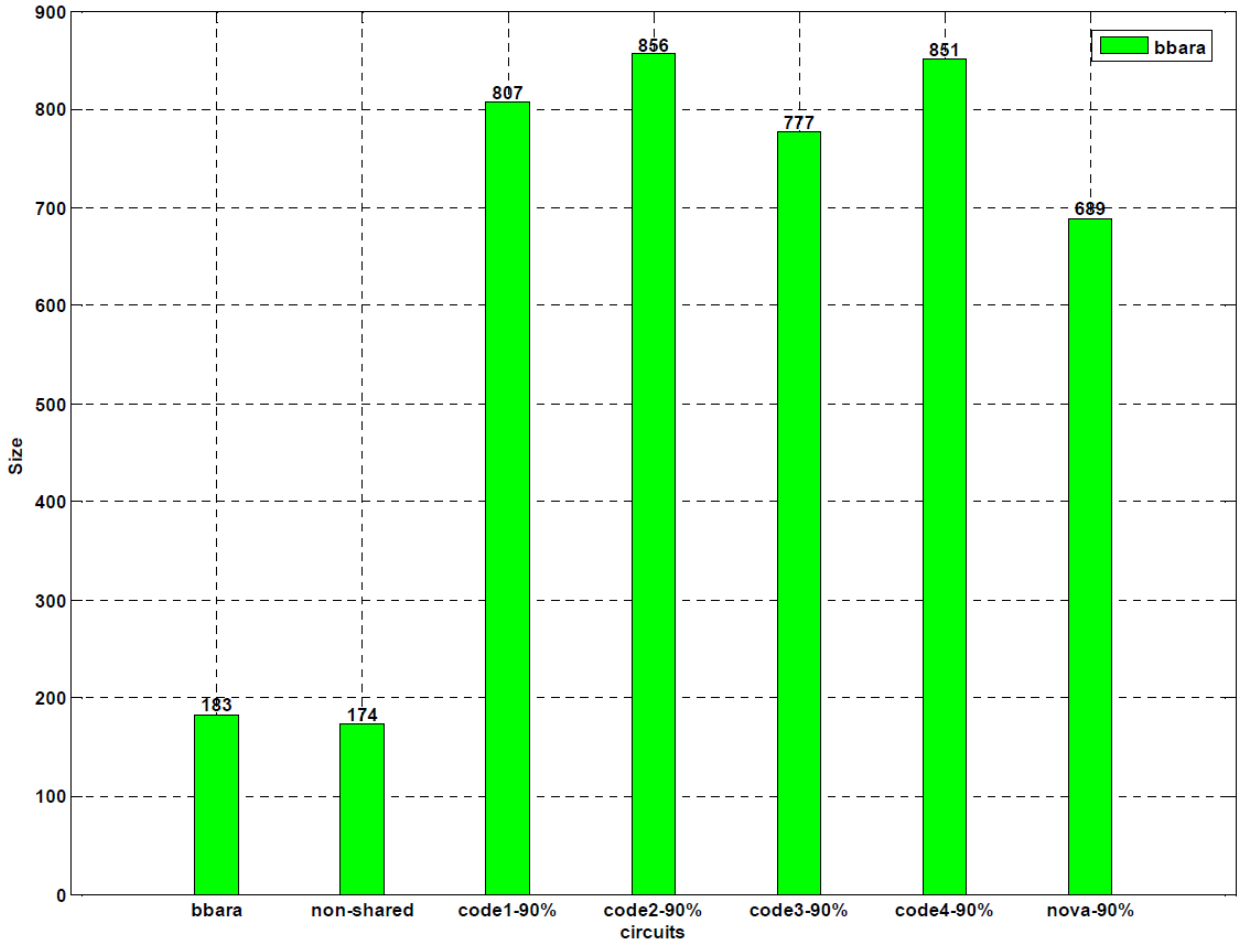


Figure 6.39: Size of bbara experiments.

an area optimized sequential circuit using nova to add another layer of protection using state redundancy, the resulting area overhead remains minimal.

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 sequential circuits at the design level, namely finite state machine (FSM), to increase its susceptibility to radiation induced transient faults or temporal soft errors.

We have introduced a novel idea, presented as Algorithm 1, to increase sequential circuit reliability and hence fault tolerance by introducing redundant equivalent states to the states with high probability of occurrence in sequential

circuit behavioral machine. To maintain the same operation of the unprotected FSM, the newly added redundant states have the same input, next state, and output as the original states. Other states with low probability are kept without adding redundancy. This way the area overhead is kept minimal. The analysis of the problem suggests that the hamming distance between protected states codes should be at least 3 to guarantee no overlapping between the redundant states codes. According to our knowledge, none has presented the introduction of redundant states for states with high probability of occurrence at the FSM level. This idea introduces a new class of methods that tackle the soft error effect in early stages, namely at the design stage.

The experimental results show that the best failure rate reduction is found to be 91% on average when 100% state probability coverage protection is chosen for up to 30 injected faults. However, the area overhead is about 13 times the original circuit and all the states must be protected by redundant states. We avoid this by protecting states yielding 90% (50% or 25%) state probability coverage with area overhead about 4 times the original size (2.5 time for 50% or by adding 1.8 times the original area for 25%). The number of states that should be protected in 90% state probability coverage is 40% of the states in the FSM on average (21% for 50% state probability coverage or 9% for 25%).

Next, state assignment has been explored and found to have a minimal impact on soft error tolerance of sequential circuits. Hence, another technique has been proposed to enhance reliability to a sequential circuit that has a specific state

assignment optimizing a certain criteria such as area. The technique, presented as Algorithm 2 starts from a state assignment or state encoding that optimizes a certain criteria such as area or power, and adds fault tolerance on top of it so that the optimization is kept. An optimized sequential circuit for area using nova can be enhanced by adding a layer of protection using state redundancy by adding extra bits to the left of chosen protected states encodings that have high probability of occurrence. As the original encoding of the protected states optimizes area on one hand, the extra added bits, which guarantee a hamming distance of 3 between the protected states, result in fault tolerance property on the other hand.

Since we started from a minimum point of area overhead which results from the original state encoding, the resulting area overhead in the protected circuit remains minimal. This is explained by the fact that cubes from the original FSM are actually either kept the same, result in a reduced form of them (more literals are added), split into two or more cubes, or/and additional cubes are added to cover the redundant states. We conclude that regardless of the encoding of states, the failure rate of protecting the same states is similar for different state encoding. Choosing state assignment using nova which targets area optimization as a starting point, we can add another layer of protection using state redundancy and the resulting area overhead remains minimal.

7.1 Summary of the Contributions

The contributions of this thesis can be summarized as follows:

- Implementation of a tool for computing state probabilities for a finite state machine.

- Implementation of a tool for computing soft error reliability for sequential circuits based on Monte Carlo simulation [13]. The objective of this tool is to find the failure rate of a sequential circuit as more faults are observed in the circuit. The monte carlo based simulation tool has been developed using Perl [14–16] in Linux [17–20] system to assess the soft error reliability of the resulting synthesized circuits. (SIS) tool [21] has been used for synthesis and (hope) tool [22] has been used for simulation purposes.

- Development and implementation of an algorithm, Algorithm 1, to enhance reliability of sequential circuits based on adding redundant states at the state diagram level to ensure single fault tolerance. The development of the algorithm has been done by modifying the finite state machine of a sequential circuit by adding the minimum necessary equivalent redundant states to selected states for which soft error tolerance is desired. Also, the minimum number of bits used in state encoding that ensure fault tolerance of single faults has been computed. It was done using C# [23].

- Evaluation of the proposed soft error tolerant design algorithm in terms of circuit reliability and other parameters including area and comparison with

existing techniques. A tool has been developed for assigning codes to states according to certain developed heuristics. (SIS) tool has been used in the synthesis process and the developed monte carol based simulation tool has been used to assess soft error tolerance of synthesized circuits.

- Investigating the impact of state assignment on sequential circuits reliability which lead to the implementation of Algorithm 2 that increases reliability of an optimized sequential circuit.

- Development and implementation an algorithm, Algorithm 2, for state assignment to enhance soft error tolerance on top of an area optimized design.

- Evaluation of the proposed state assignment for soft error tolerance in terms of soft error reliability and area.

7.2 Future Work

This work led to many ideas and many possible future work. They are:

- Extending the idea of adding redundancy using probability calculation, that enhances fault tolerance, to combinational circuits. This can be achieved by taking advantage of knowing the likelihood of an intermediate line in a digital circuit to has a value of 1 or 0. For example, the output of an AND-gate is 3/4 of the time 0; so it is better to protect the 0 of that line more than the 1.

- Developing an algorithm that enhances reliability for a power optimized sequential circuit. By starting from a state assignment that gives lower power consumption to sequential circuits and applying it to Algorithm 2 as input, it is expected that the power consumption will also be kept minimal.

References

[1] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, April 1965.

[2] M. Butts, A. DeHon, and S. C. Goldstein, "Molecular electronics: devices, systems and tools for gigagate, gigabit chips," in *ICCAD '02: Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, (New York, NY, USA), pp. 433–440, ACM, 2002.

[3] T. N. A. Bachtold, P. Harley, and C. Dekker, "Logic circuits with carbon nanotube transistors," *Science Express*, 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, "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, February 1994.

[7] M. M. Mano and C. R. Kime, *logic and computer design fundementals*. Prentice Hall, 2001.

[8] P. Ashar, S. Devadas, and R. Newton, *Sequential Logic Synthesis*. Kluwer Academic, 1992.

[9] M. Perkowski, *Design, Test and Verification of Sequential Circuits*. Marek Perkowski.

[10] P. Mavis, D.; Eaton, "Seu and set mitigation techniques for fpga circuit and configuration bit storage design," in *International Conference on Military and Aerospace Applications of Programmable Logic Devices, MAPLD*, September 2000.

[11] D. G. Mavis and P. H. Eaton, "Temporally redundant latch for preventing single event disruptions in sequential integrated circuits," technical report, Mission Research, September 1998.

[12] D. Rossi, M. Omana, F. Toma, and C. Metra, "Multiple transient faults in logic: an issue for next generation ics?," in *Defect and Fault Tolerance in VLSI Systems, 2005. DFT 2005. 20th IEEE International Symposium on*, pp. 352 – 360, oct. 2005.

[13] M. H. Kalos and P. A. Whitlock, *Monte Carlo Methods.* John Wiley and Sons, 1986.

[14] R. Schwartz, T. Phoenix, and B. d foy, *Learning Perl (4th edition).* O'Reilly & Associates, 2005.

[15] S. Srinivasan, *Advanced Perl Programming.* O'Reilly Media, 1997.

[16] T. N. Harvey Deitel, Paul Deitel and McPhie, *Perl How to Program.* Prentice Hall, 2001.

[17] W. R. Stevens, *Advanced Programming in the UNIX Environment.* Addision Wesly, 1993.

[18] Mitchell, Oldham, and Samuel, *Advanced Linux Programming.* New Riders, 2001.

[19] A. Robbins and N. H. F. Beebe, *Classic Shell Scripting.* O'Reilly Media, 2005.

[20] L. Manual, *http://linuxmanpages.com/.*

[21] E. Sentovich, K. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Sis: A system for sequential circuit synthesis," Tech. Rep. UCB/ERL M92/41, EECS Department, University of California, Berkeley, 1992.

[22] H. K. Lee and D. S. Ha, "Hope: an efficient parallel fault simulator for synchronous sequential circuits," in *DAC '92: Proceedings of the 29th*

ACM/IEEE Design Automation Conference, (Los Alamitos, CA, USA), pp. 336–340, IEEE Computer Society Press, 1992.

[23] J. L. Harvey Deitel, Paul Dietel and et al., *C# How to Program*. Prentice Hall, 2008.

[24] M. L. Bushnell and V. D. Agrawal, *Essentials of electronic testing for digital, memory, and mixed-signal VLSI circuits*. Springer, 2000.

[25] M. S. MacDiarmid Preston and et al., "Reliability toolkit: Commercial practices edition," *Reliability Analysis Center and Rome Laboratory*, pp. 35–39, 1995.

[26] C. E. Ebeling, *An Introduction to Reliability and Maintainability Engineering*. McGraw-Hill Companies, 1997.

[27] T. P. Ma and P. V. Dressendorfer, *Ionizing Radiation Effects In Mos Devices And Circuits*. John Wiley & Sons, Incorpo, 1989.

[28] S. Mitra, N. Seifert, M. Zhang, Q. Shi, and K. Kim, "Robust system design with built-in soft-error resilience," *Computer*, vol. 38, pp. 43 – 52, feb. 2005.

[29] N. Miskov-Zivanov and D. Marculescu, "Soft error rate analysis for sequential circuits," in *DATE '07: Proceedings of the conference on Design, automation and test in Europe*, (San Jose, CA, USA), pp. 1436–1441, EDA Consortium, 2007.

[30] Tezzaron, "Soft errors in electronic memory," *A White Paper http://www.tezzaron.com/about/papers/soft_errors_1_1_secure.pdf*, Jan. 2004.

[31] D. P. Siewiorek and R. S. Swarz, *Reliable computer systems, design and evaluation.* Digital Press, 2nd ed., 1992.

[32] A. Avizienis, "Design of fault-tolerant computers," in *Joint Computer Conference of AFIPS*, pp. 733 – 743, 1967.

[33] P. K. Lala, *Self-checking and fault-tolerant digital design.* Morgan Kaufmann, 2nd ed., 1992.

[34] M. Hartmann, *Evolution of fault and noise-tolerant digital circuits, Doctoral Dissertation.* Norwegian University of Science and Technology, April 2005.

[35] A. Namazi and M. Nourani, "Reliability analysis and distributed voting for nmr nanoscale systems," in *Design and Test Workshop, 2007. IDT 2007. 2nd International*, pp. 130 –135, 16-18 2007.

[36] C. He, M. Jacome, and G. de Veciana, "A reconfiguration-based defect-tolerant design paradigm for nanotechnologies," *Design Test of Computers, IEEE*, vol. 22, pp. 316 – 326, july-aug. 2005.

[37] S. Krishnamohan and N. Mahapatra, "A highly-efficient technique for reducing soft errors in static cmos circuits," in *Computer Design: VLSI in Computers and Processors, 2004. ICCD 2004. Proceedings. IEEE International Conference on*, pp. 126 – 131, oct. 2004.

[38] S. Krishnamohan and N. R. Mahapatra, "Analysis and design of soft-error hardened latches," in *GLSVLSI '05: Proceedings of the 15th ACM Great Lakes symposium on VLSI*, (New York, NY, USA), pp. 328–331, ACM, 2005.

[39] K. Hass, J. Gambles, B. Walker, and M. Zampaglione, "Mitigating single event upsets from combinational logic," in *Proceedings of 7th NASA Symp on VLSI design*, 1998.

[40] Y. Komatsu, Y. Arima, T. Fujimoto, T. Yamashita, and K. Ishibashi, "A soft-error hardened latch scheme for soc in a 90 nm technology and beyond," in *Custom Integrated Circuits Conference, 2004. Proceedings of the IEEE 2004*, pp. 329 – 332, 3-6 2004.

[41] T. Calin, M. Nicolaidis, and R. Velazco, "Upset hardened memory design for submicron cmos technology," *Nuclear Science, IEEE Transactions on*, vol. 43, pp. 2874 –2878, dec 1996.

[42] P. Hazucha, T. Karnik, S. Walstra, B. Bloechel, J. Tschanz, J. Maiz, K. Soumyanath, G. Dermer, S. Narendra, V. De, and S. Borkar, "Measurements and analysis of ser-tolerant latch in a 90-nm dual-vt cmos process," *Solid-State Circuits, IEEE Journal of*, vol. 39, pp. 1536 – 1543, sept. 2004.

[43] W. Stallings, *Data and Computer Communications*. Prenice Hall, 7th ed., 2004.

[44] I. Levin, V. Ostrovsky, and S. Ostanin, "Self-healing ability of sequential circuits," in *Electrical and Electronics Engineers in Israel, 2002. The 22nd Convention of*, pp. 114 – 116, dec. 2002.

[45] J. M. Berger, "A note on an error detection code for asymmetric channels," *Information and Control*, vol. 4, p. 6873, March 1961.

[46] J. E. Smith, "On separable unordered codes," *Computers, IEEE Transactions on*, vol. C-33, pp. 741 –743, aug. 1984.

[47] J. Smith and G. Metze, "Strongly fault secure logic networks," *Computers, IEEE Transactions on*, vol. C-27, pp. 491 –499, june 1978.

[48] H. Asadi and M. Tahoori, "Soft error modeling and protection for sequential elements," in *Defect and Fault Tolerance in VLSI Systems, 2005. DFT 2005. 20th IEEE International Symposium on*, pp. 463 – 471, oct. 2005.

[49] G. Asadi and M. Tahoori, "An analytical approach for soft error rate estimation in digital circuits," in *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*, pp. 2991 – 2994 Vol. 3, may 2005.

[50] H. Asadi and M. B. Tahoori, "Soft error derating computation in sequential circuits," in *ICCAD '06: Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, (New York, NY, USA), pp. 497–501, ACM, 2006.

[51] H. Asadi, M. Tahoori, and C. Tirumurti, "Estimating error propagation probabilities with bounded variances," in *Defect and Fault-Tolerance in VLSI Sys-*

tems, 2007. DFT '07. 22nd IEEE International Symposium on, pp. 41 –49, sept. 2007.

[52] J. P. Hayes, I. Polian, and B. Becker, "An analysis framework for transient-error tolerance," in *VTS '07: Proceedings of the 25th IEEE VLSI Test Symposium*, (Washington, DC, USA), pp. 249–255, IEEE Computer Society, 2007.

[53] M. R. Choudhury and K. Mohanram, "Accurate and scalable reliability analysis of logic circuits," in *DATE '07: Proceedings of the conference on Design, automation and test in Europe*, (San Jose, CA, USA), pp. 1454–1459, EDA Consortium, 2007.

[54] J. Han, E. Taylor, J. Gao, and J. Fortes, "Faults, error bounds and reliability of nanoelectronic circuits," in *ASAP '05: Proceedings of the 2005 IEEE International Conference on Application-Specific Systems, Architecture Processors*, (Washington, DC, USA), pp. 247–253, IEEE Computer Society, 2005.

[55] C. A. Lisboa, M. I. Erigson, and L. Carro, "System level approaches for mitigation of long duration transient faults in future technologies," in *ETS '07: Proceedings of the 12th IEEE European Test Symposium*, (Washington, DC, USA), pp. 165–172, IEEE Computer Society, 2007.

[56] Walpole, Myers, and Myers, *Probability and Statistics for Engineering and Scientists*. Prentic Hall, 6th ed., 1998.

[57] T.-L. Chou and K. Roy, "Statistical estimation of sequential circuit activity," in *ICCAD '95: Proceedings of the 1995 IEEE/ACM international conference*

on Computer-aided design, (Washington, DC, USA), pp. 34–37, IEEE Computer Society, 1995.

[58] F. N. Khan, *Finite State Machine Encoding/State Assignment for Low Power, Reduced Area and Increased Testability using Iterative Algorithms.* KFUPM: M.Sc, 2005.

[59] K. Parker and E. McCluskey, "Probabilistic treatment of general combinational networks," *Computers, IEEE Transactions on*, vol. C-24, pp. 668 – 670, june 1975.

[60] G. Hachtel, E. Macii, A. Pardo, and F. Somenzi, "Probabilistic analysis of large finite state machines," in *Design Automation, 1994. 31st Conference on*, pp. 270 – 275, 6-10 1994.

[61] J. Savir, G. S. Ditlow, and P. H. Bardell, "Random pattern testability," *Computers, IEEE Transactions on*, vol. C-33, pp. 79 –90, jan. 1984.

[62] M. Nicolaidis, "On-line testing for vlsi: state of the art and trends," *Integration, the VLSI Journal*, vol. 26, no. 1-2, pp. 197 – 209, 1998.

[63] D. Marculescu, R. Marculescu, and M. Pedram, "Trace-driven steady-state probability estimation in fsms with application to power estimation," in *Design, Automation and Test in Europe, 1998., Proceedings*, pp. 774 –779, 23-26 1998.

[64] G. Hachtel, M. Hermida, A. Pardo, M. Poncino, and F. Somenzi, "Re-encoding sequential circuits to reduce power dissipation," in *Computer-Aided*

Design, 1994., IEEE/ACM International Conference on, pp. 70 –73, 6-10 1994.

[65] A. El-Maleh, S. Sait, and F. Nawaz Khan, "Finite state machine state assignment for area and power minimization," in *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on*, p. 4 pp., 0-0 2006.

[66] C.-Y. Tsui, J. Monteiro, M. Pedram, S. Devadas, A. Despain, and B. Lin, "Power estimation methods for sequential logic circuits," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 3, pp. 404 –416, sep 1995.

[67] G. Hachtel, E. Macii, A. Pardo, and F. Somenzi, "Markovian analysis of large finite state machines," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 15, pp. 1479 –1493, dec 1996.

[68] A. Papoulis, *Random Variables and Stochastic Processes*. McGraw-Hill, 1984.

[69] Z. Kohavi., *Switching and Finite Automata Theory*. McGraw-Hill, 2nd ed., 1978.

[70] J. Hartmanis, "On the state assignment problem for sequential machines. i," *Electronic Computers, IEEE Transactions on*, vol. EC-10, pp. 157 –165, june 1961.

[71] R. E. Stearns and J. Hartmanis, "On the state assignment problem for sequential machines ii," *Electronic Computers, IEEE Transactions on*, vol. EC-10, pp. 593 –603, dec. 1961.

[72] H. A. Curtis, "Multiple reduction of variable dependency of sequential machines," *J. ACM*, vol. 9, no. 3, pp. 324–344, 1962.

[73] M. Cassel and F. L. Kastensmidt, "Evaluating one-hot encoding finite state machines for seu reliability in sram-based fpgas," *IEEE International On-Line Testing Symposium*, vol. 0, pp. 139–144, 2006.

[74] W. M. Aly, "Solving the state assignment problem using stochastic search aided with simulated annealing," *American J. of Engineering and Applied Sciences*, vol. 2, no. 4, pp. 710–714, 2009.

[75] D. S. Hochbaum, ed., *Approximation algorithms for NP-hard problems*. Boston, MA, USA: PWS Publishing Co., 1997.

[76] I. Ahmad and M. Dhodhi, "State assignment of finite-state machines," *Computers and Digital Techniques, IEE Proceedings* -, vol. 147, pp. 15 –22, jan 2000.

[77] S. Devadas and A. Newton, "Exact algorithms for output encoding, state assignment, and four-level boolean minimization," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 10, pp. 13 –27, jan 1991.

[78] B. Eschermann, "State assignment for hardwired vlsi control units," *ACM Comput. Surv.*, vol. 25, no. 4, pp. 415–436, 1993.

[79] G. De Micheli, "Symbolic design of combinational and sequential logic circuits implemented by two-level logic macros," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 5, pp. 597 – 616, october 1986.

[80] T. Villa and A. Sangiovanni-Vincentelli, "Nova: state assignment of finite state machines for optimal two-level logic implementations," in *DAC '89: Proceedings of the 26th ACM/IEEE Design Automation Conference*, (New York, NY, USA), pp. 327–332, ACM, 1989.

[81] G. De Micheli, R. Brayton, and A. Sangiovanni-Vincentelli, "Optimal state assignment for finite state machines," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 4, pp. 269 – 285, july 1985.

[82] P. Ashar, S. Devadas, and A. R. Newton, *Sequential Logic Synthesis*. Norwell, MA, USA: Kluwer Academic Publishers, 1992.

[83] S. Devadas, H.-K. Ma, A. Newton, and A. Sangiovanni-Vincentelli, "Mustang: state assignment of finite state machines targeting multilevel logic implementations," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 7, pp. 1290 –1300, dec 1988.

[84] X. Du, G. Hachtel, and P. Moceyunas, "Muse: a multilevel symbolic encoding algorithm for state assignment," in *System Sciences, 1990., Proceedings of the*

Twenty-Third Annual Hawaii International Conference on, vol. i, pp. 367 – 376 vol.1, 2-5 1990.

[85] T. Villa., "Constrained encoding in hypercubes: Algorithms and applications to logical synthesis," *Memo UCB/ERL M87137, Univ. California, Berkeley*, May 1987.

[86] T. Villa., "nova, users manual," *Univ. California, Berkeley*, Oct 1988.

[87] J. Han, J. Gao, Y. Qi, P. Jonker, and J. A. B. Fortes, "Toward hardware-redundant, fault-tolerant logic for nanoelectronics," *IEEE Des. Test*, vol. 22, no. 4, pp. 328–339, 2005.

Vita

- General Information:

  - Ayed Saad Al-Qahtani.
  - Male.
  - Saudi Arabian.

- Addresses and Communication:

  - P.O.Box 13340, Riyadh 11493, Kingdom of Saudi Arabia.
  - ayedsaad@gmail.com

    ayedsaad@hotmail.com
  - Mobile: 0555993338.

- Education & Qualification:

  - Bachelor of Computer & Information Sciences, King Saud University (KSU), Computer Engineering Department, excellent grade, first honor class, 2007.

  - Master of Computer & Information Sciences, King Fahad University for Petroleum & Minerals (KFUPM), Computer Engineering Department, excellent grade, first honor class, 2010.

- Work Experience:
  - as R&D Engineer & Testing Engineer at Advanced Electronic Company (AEC) in Riyadh, Saudi Arabia, for 3 months.
  - as TA in King Saud University (KSU), Computer Engineering Department in Riyadh, Saudi Arabia, for 1.5 year.
  - as Smart Card Developer in Al-ELM Information Security Company in Riyadh, Saudi Arabia, for 3 months.
  - as Consultant in Al-ELM Information Security Company in Riyadh, Saudi Arabia, for 1.5 year.

- Publications and Researches:
  - "Asymptotic Behavior of Networked Control System (NCS)" 18th National Computer Conference (NCC18), Riyadh, 2007
  - "Triple-A: Secure RGB Image Steganography Based on Randomization", AICCSA-2009 - The 7th ACS/IEEE International Conference on Computer Systems and Applications, Pages: 400-403, Rabat, Morocco, 10-13 May 2009.

- Skills, Interests and Activities:

– Communication, team work, problem solving, traveling, creating new friendships, building networks, football, swimming, weight training & personal fitness, participating in solving new computer problem challenges.