Soft Error Analysis and Mitigation at High Abstraction Levels

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Liang Chen

aus Henan, China

Tag der mündlichen Prüfung: June 19, 2015

Referent: Prof. Dr. Mehdi Baradaran Tahoori, KIT

Korreferent: Dr. Dan Alexandrescu, iRoC Technologies

Acknowledgments

Completing the PhD is a long expedition for me, during which I underwent unforgettable life experience: from daunting and discouraging periods after bitter frustrations on research and publications, to the comforting and reassuring moments after fruitful project progress and final acknowledgment from the research community. Standing at the destination point and looking back at the old times, I want to express my deepest gratitude to all the people accompanying me in this journey.

First and foremost I want to give the sincere thanks to my advisor Prof. Mehdi Tahoori. During the more than 4 years PhD period, his visionary advice on my research directions, his continuous care of my research progress as well as his patient efforts to improve my research capabilities are the prerequisites for my successful completeness of this dissertation today. In addition, his encouragement after research difficulties and paper rejections, the valuable opportunities he provided for communications with scientific experts as well as industry leaders during international conferences or workshops, are other indispensable factors for my professional and personal development. In particular, during the internship at iRoC I met Dr. Dan Alexandrescu and had impressive working experience with him. His extraordinary expertise in my research period, the inspiring discussion with him on the cutting edge industry progress, and the efforts he spent as the second advisor during the graduation progress are all highly appreciated.

Furthermore, I want to thank all the colleagues in our Chair of Dependable Nano Computing (CDNC). Their responsible and kind-hearted help in both working and personal aspects facilitated my research activities in CDNC, as well as my living experience in Karlsruhe. Among them I would like to give special gratitude to Mojtaba Ebrahimi, with whom I had very close research collaborations in the past years. His inspiring advices during idea formulation, his consistent help in the implementations, paper writing and polishing have made significant contributions to my PhD achievements.

In addition, I gratefully acknowledge the funding sources that made my PhD work possible: the DFG SPP1500 Priority Program "PERICES: Providing Efficient Reliability in Critical Embedded Systems".

I am also very grateful to my friend and senior Dr. Bing Li from Technical University Munich, who gave me essential advices and encouragement to go through the toughest period during my PhD study. Last but not least, I want to thank my parents and sister who are accompanying me with endless love all the time. No matter what happens, they are always supporting me and standing on my side, which is the root source of all my power and passion in the life.

Liang Chen
Haid-und-Neu Str. 62
76131 Karlsruhe

Hiermit erkläre ich an Eides statt, dass ich die von mir vorgelegte Arbeit selbstständig verfasst habe, dass ich die verwendeten Quellen, Internet-Quellen und Hilfsmittel vollständig angegeben haben und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen - die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Karlsruhe, Juli 2015

_____

Liang Chen

ABSTRACT

With the rapid advance of semiconductor technology in the past several decades, computing systems already become indispensable parts of our society. The main driving forces behind the prevalent usage of these systems are their boosting performance and continuous integration of sophisticated functionalities. However, alongside with the aggressive downscaling of the transistor feature size in these systems, the current and future technologies are facing serious reliability challenges, which mandate a new paradigm to take reliability into consideration during the entire system design flow to achieve cost-efficient reliable computing.

One of the major reliability issues is radiation-induced soft error, which is a type of transient fault during the system operation, and it is caused by charge collection after the transistor is struck by the energetic particles from packaging materials or cosmic rays. Soft errors are not reproducible and can severely corrupt the system data integrity. The previous soft error mitigation solutions at low abstraction levels are very costly, because they stay at the end of the design stack and the useful high level application semantics for error abstraction and the corresponding error masking effects are totally ignored. In contrast, at high abstract levels there are significant potentials for cost-effective solutions, because the reliability metric are considering earlier in the design phase, and the larger flexibility in the design space exploration can be leveraged.

This thesis presents several novel and efficient techniques for soft error evaluation and mitigation at high abstract levels, i.e. from register transfer level up to behavioral algorithmic level. For error evaluation, first we perform sophisticated correlation analysis during the propagation process of multiple errors, which enables the appropriate error abstraction from logic level to register transfer level. Then at register transfer level, according to different error propagation properties, the data and control paths are carefully analyzed with analytical and formal techniques, respectively. Finally at the algorithmic level, the soft error evaluation is comprehensively performed by considering branch probabilities and functional semantics using the probabilistic model checking technique.

Based on previous detailed soft error evaluation, cost-efficient error mitigation solutions based on selective protection are investigated during the high level synthesis process. We develop a versatile reliability-aware synthesis framework regarding the allocation and binding of hardware modules (in the spatial domain), and the scheduling of behavioral operations (in the temporal domain), which can generate low-cost reliability-enhanced hardware implementations at register transfer level. This synthesis framework features the comprehensive consideration of the reliability metrics at both low and high abstraction levels, i.e. intrinsic soft error rates of hardware modules and algorithmic error vulnerabilities of behavioral variables and operations. The effectiveness of the proposed techniques is demonstrated with extensive synthesis experiments, which show that compared with the state-of-the-art solutions, our techniques can achieve much higher system reliability with the same hardware overhead, i.e. more cost-efficient reliability enhancement.

ZUSAMMENFASSUNG

Bedingt durch die schnell voranschreitende Entwicklung der Halbleitertechnologie in den vergangenen Jahrzehnten sind Computersystem mittlerweile essentielle Bestandteile unserer Gesellschaft. Der Hauptgründe für diesen Erfolg und die gängige Nutzung solcher Systeme ist die kontinuierlich wachsende Leistungsfähigkeit und die Integration von immer mehr sowie immer ausgeklügelteren Funktionen. Um dies zu ermöglichen werden die Transistordimensionen mit jeder Generation weiter verkleinert. Dies bringt allerdings auch zahlreiche Herausforderungen mit sich. So sind aktuelle und zukünftige Technologiegenerationen grossen Zuverlässigkeitsherausforderungen ausgesetzt, weshalb die Zuverlässigkeit von Computersystemen als Paradigma in allen Phasen des gesamten Entwicklungszyklus solcher Systeme beachtet werden muss.

Eine der wichtigsten Zuverlässigkeitsherausforderungen sind "Soft Errors" verursacht durch Strahlung. Dabei handelt es sich um transiente Fehler, die während des Betriebs auftreten, und durch Treffer von hochenergetischen Teilchen aus dem Verpackungsmaterial oder aus kosmischer Strahlung hervorgerufen werden. "Soft Errors" können daher nicht reproduziert werden. Darüber hinaus können sie die Datenintegrität des betroffenen Systems stark beeinträchtigen. Die bisherigen Lösungsansätze konzentrieren sich auf die untersten Abstraktionsschichten und sind daher sehr kostspielig, da in solchen Methoden "Soft Errors" nur am Ende des Designzyklus beachtet werden und nützliche Informationen von höheren Abstraktionsebenen nicht berücksichtigt werden. Dennoch haben höhere Abstraktionsebenen ein grosses Potential um zu einer effektiven Problemlösung beizutragen, da die Zuverlässigkeitsprobleme bereits in frühen Phasen des Entwicklungszyklus betrachtet werden können und zudem ein grösserer Suchraum für Lösungsansätze besteht.

In dieser Dissertation werden neue Ansätze zur Modellierung und zur Problemlösung für "Soft Errors" auf höheren Abstraktionsebenen (von Register-Transfer-Ebene bis zur Algorithmen-Ebene) vorgestellt. Um eine geeignete Abstraktion der Fehlermechanismen zu erreichen, müssen dazu die Korrelationen von diversen Fehlerursachen auf Logikebene untersucht werden. Anschliessend folgt eine weitere Abstraktion auf Register-Transfer-Ebene, auf der die Daten- und Kontrollpfade mit Hilfe von analytischen sowie formalen Methoden analysiert werden. Schlussendlich wird eine "Soft Error" Evaluierung auf noch höheren Abstraktionsebenen unter der Berücksichtigung der Verhaltensbeschreibung des Designs durchgeführt. Hierzu wird ein probabilitischer Model-Checker-Ansatz entwickelt, der Sprungwahrscheinlichkeiten und semantischen Aussagen des Designs in die Analyse miteinbezieht.

Basierend auf der detaillierten "Soft Error" Analyse können dann kostengünstige Ansätze entwickelt werden um die Fehlerrate zu verringen, in dem selektive Schutzmöglichkeiten währen der High-Level Synthese ausgenutzt werden. Zu diesem Zweck haben wir eine Synthese-Plattform auf Register-Transfer-Ebene entwickelt, die die Zuverlässigkeit der Schaltung berücksichtigt. Mit dieser Plattform ist es möglich, eine optimale Strategie hinsichtlich der Zuverlässigkeit für die Allokierung und Bindung der Hardware-Ressourcen (örtliche Domäne) sowie für das Scheduling der Operationen (zeitlich Domäne) zu finden, um auf diese Weise die Zuverläs-

sigkeit der Schaltung zu verbessern. Dafür baut die Plattform auf ausführlich beschriebenen Zuverlässigkeitsmetriken auf. Die Effektivität der vorgestellten Ansätze während der High-Level Synthese wird in ausführlichen Experimenten beleuchtet und mit State-of-the-Art Lösungen verglichen. Diese Vergleiche zeigen dabei, dass unsere Methoden wesentlich effizienter die Zuverlässigkeit verbessern können, d.h. mit den gleichen Kosten kann die Zuverlässigkeit des Systems deutlich gegenüber gängigen Lösungen gesteigert werden.

Contents

Glossary

A | **B** | **C** | **D** | **E** | **F** | **H** | **I** | **L** | **M** | **N** | **P** | **R** | **S** | **T** | **V**

A

ACE Architecturally Correct Execution.

ADPCM Adaptive Differential Pulse Code Modulation.

ALU Arithmetic Logic Unit.

AVF Architectural Vulnerability Factor.

B

BB Basic Block.

BISER Built-in Soft Error Resilience.

C

CC Correlation Coefficient.

CCM Correlation Coefficient Method.

CDFG Control and Data Flow Graph.

CEP Correlated Error Propagation.

CFI Control Flow Intensive.

CMOS Complementary Metal-Oxide-Semiconductor.

D

DFG Data Flow Graph.

DFI Data Flow Intensive.

DFS Depth First Search.

DICE Dual Interlock Storage Cells.

DTR Data Type Reduction.

E

ECC Error Correction Code.

EPF Error Propagation Function.

EPP Error Propagation Probability.

F

FI Fault Injection.

FIT Failure in Time.

FPE Failures per Execution.

FSM Finite State Machine.

FU Functional Unit.

H

HLS High Level Synthesis.

I

IC Instruction Cache.
ILP Integer Linear Programming.

L
LLVM Low Level Virtual Machine.
LP Linear Programming.

M
MBG Mapping and Binding Graph.
MBU Multi Bit Upset.
MC Monte-Carlo simulation.
MCU Multi Cell Upset.
MDP Markov Decision Process.
MPSoC Multi-Processor System on Chip.
MTTF Mean Time To Failure.
MUX Multiplexer.

N
NP Nondeterministic Polynomial.

P
PCC Pearson product-moment Correlation Coefficient.
PI Primary Input.
PMC Probabilistic Model Checking.
PO Primary Output.
PRISM A probabilistic model checker.

R
RAM Random Access Memory.
RTL Register-Transfer Level.

S
SAT Boolean Satisfiability.
SBU Single Bit Upset.
SDG Signal Dependency Graph.
SEE Single Event Effects.
SEL Single Event Latchup.
SER Soft Error Rate.
SET Single Event Transient.
SEU Single Event Upset.
SFI Statistical Fault Injection.
SFR System Failure Rate.
SOI Silicon-On-Insulator.
SP Signal Probability.
SPICE Simulation Program with Integrated Circuit Emphasis.

T
TCAD Technology Computer-Aided Design.

TMR Triple Modular Redundancy.

TPP Time-multiplexed PIs Partitioning.

V

VAS Verification Assisted Simplification.

VCD Value Change Dump.

VLSI Very-Large-Scale Integration.

List of Figures

List of Tables

1 Introduction

With the rapid development of semiconductor industry in the past several decades, computing systems nowadays become prevalent and indispensable parts of the human society. A variety of application domains of the computing systems range from high-end mainframes and mission-critical applications, such as aerospace satellites and civil aircraft, to the daily consumer electronics such as laptops, smart phones and tablets. The primary driving force behind the astonishing growth of computing system is the boosting performance and continuous integration of sophisticated functionalities in the Very-Large-Scale Integration (VLSI) chips. According to Moore's law, which was first described by Gordon E. Moore in 1965 [7] and later proved by the semiconductor industry, the number of transistors per integrated circuit chip doubles approximately every two years. The corresponding aggressive scaling of the transistor size, however, is facing serious reliability challenges, particularly in the nanoscale technology era. The computing systems become more susceptible to the manufacturing and environmental imperfections, e.g. process variations and runtime variations. These challenges lead to a new paradigm considering reliability as an additional design metric or constraint in the design space exploration.

One of the major reliability issues is the radiation-induced soft error, which is transient fault arising from the strike of energetic particles, such as alpha particles in packaging material or neutrons from the atmosphere. When the particles with high energy pass through the semiconductor device, e.g. Metal-Oxide-Semiconductor Field-Effect Transistor, and generate electron and hole pairs, the electric charge can be collected by the source and drain nodes of the transistors. If the accumulated charge exceeds a specific threshold (the so-called "critical charge"), the strike may invert the state of a logic component, such as a memory cell, latch or gate, thereby introducing a transient fault into the circuit operation. These errors are called *"soft"* because rather than permanent malfunction of the impaired circuit, they have the intermittent and transient characteristics. Compared with the permanent faults (i.e. hard errors), soft errors are not reproducible and can occur even more often and severally corrupt the system data integrity [8].

For many years soft error was regarded as a reliability issue only for the microelectronics in aerospace applications, because the density of high energy particles in space is several orders of magnitude larger than that at terrestrial level [9]. However, starting from 1980s this kind of radiation-induced transient error has been demonstrated to be present in computing system at sea level. The first report on soft errors due to the alpha particles in the contaminated packing materials was from Intel company in 1978 [10], while the soft errors arising from cosmic radiation were predicated by Ziegler and Lanford in 1979 [11], and first reported by IBM corporation in 1984 [12].

Subsequently the soft error issue has become one of the major reliability concerns in semiconductor industry [3]. With the transistor feature size continuously shrinking, the critical

charge is decreasing and a particle strike is more likely to cause soft error. Even the particles of lower energy, which have much larger amount than the high energy ones, can generate sufficient charge to flip the circuit state. Aside from the traditional single bit error, the multiple bit upsets or transients are already coming into the picture [13]. Furthermore, although the Soft Error Rate (SER) per device, the rate at which soft errors appear in a device for a given environment, is predicted to remain roughly constant over several technology nodes [14], the continuously increasing integration of transistors per chip (i.e. Moore's law) makes the system soft error rate approximately double with the advance of each technology node [15]. In consequence, soft error is emerging as a significant reliability obstacle to the microprocessors manufactured in nano technologies, and has already caused significant commercial loss in recent years [12, 16, 17]. Therefore, soft error mitigation is becoming indispensable for an increasing number of application domains at the ground level such as networking, servers, medical, and automotive electronics.

Traditionally, the high-end computing systems always used redundant copies of hardware components to detect and recover from errors. However, as in nanoscale era the soft error threat spreads to the mainstream commodity computing markets, such kind of massive-redundancy-based solutions have been prohibitively expensive and difficult to justify due to the high-volume production and stringent constraints on the cost and power budgets. Although the efficient coding techniques, such as parity and Error Correction Code (ECC), can be employed to protect the regular memory blocks, they are not applicable to irregular sequential flip-flops and combinational gates, which already become the dominant contributors to the system soft error rate. The necessity to find cost-efficient reliability enhancement solutions has driven the accurate quantitative evaluation of soft errors and the successive selective protection, which can mitigate the soft error effects in an efficient way.

In the previous work on soft error evaluation and mitigation, most of the researchers focused at the low abstraction levels. At device level the interactions among striking particles and the silicon atoms in the semiconductor materials are analyzed in details, and at circuit level the generation and propagation of radiation-induced transient current pulses are simulated by SPICE. These fundamental analysis are essential and can obtain very accurate intrinsic soft error rates. However, they can not take the error masking effects at architectural or application levels into consideration, when the occurred errors at low levels propagate upwards in the abstraction stacks. Furthermore, with the increased underlying complexity of future computing system (with larger number of cores, heterogeneous integration of hardware components), appropriate error abstraction ascending from low abstraction levels, modular evaluation and simultaneous optimization on reliability and other traditional design metrics, are becoming imperative to explore the cost-effective reliable designs.

1.1 Scope and Contributions

In a typical digital system design flow as shown in Figure 1.1, the design process starts from a system specification and involves various synthesis and optimization phases at different levels of abstraction. At each level the design functionalities are equivalent but the structural components are different. The more refinements are performed, the smaller the component granularity will become. At the end of this design flow, the physical layout of the design is

finalized after rigorous verification, then it will be taped out to the manufacturing process in a semiconductor foundry to obtain the real chips. During this entire design flow, various automatic refinement processes - *synthesis* can be preformed to realize the design description with components from the lower abstraction levels. In addition, to meet the final system requirements without too many design iterations, the design metrics of the hardware components such as power, area and reliability, which are typically *abstracted* from low levels, are also indispensable inputs of the synthesis process.

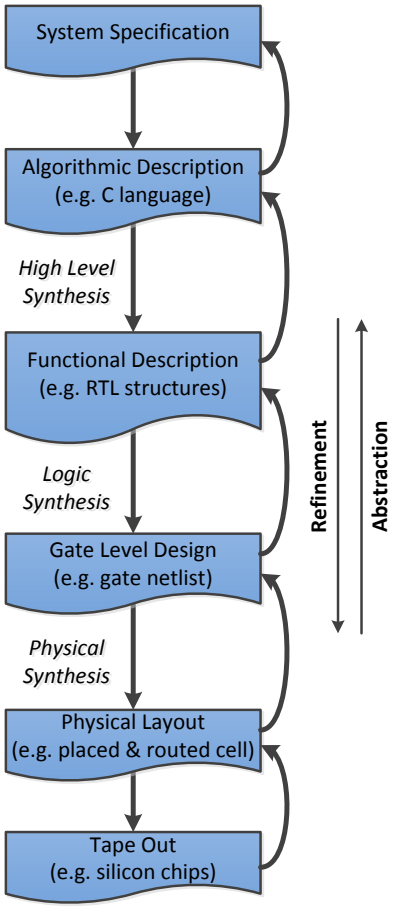


Figure 1.1: A typical digital system design flow

Regarding reliability analysis and improvement during the design flow, in contrast to previous techniques focusing mainly on gate and lower levels, in this thesis we deal with the problem of soft error evaluation and mitigation at higher abstraction levels. The objective is to accurately evaluate the contribution of individual hardware component regarding SER, and then alleviate the error impact in a cost-efficient way – selective protection. To achieve this goal, the detailed analysis of error propagation at each abstraction level (e.g. logic level, register transfer level or algorithmic level), and also error abstraction across the level boundaries need to be performed properly. Based on the detailed error rate evaluation, error mitigation by selective protection scheme, which just partially protects or hardens the most vulnerable components to achieve maximum reliability enhancement with minimum overhead, can be performed at various abstraction levels. Nevertheless, compared with low level solutions, the optimization starting from specifications at higher levels of abstraction can obtain more benefits, because it leverages the behavioral semantics which are ignored by low level techniques, and hence can provide larger flexibility in the design space exploration.

In particular, the novel contributions of this thesis, visualized on the well-known Y-chart [18] in Figure 1.2, are described in details as following:

- **Logic level soft error evaluation with correlation analysis**.

  Single particle strike on the transistors at device level can result in multiple correlated bit flips during the error propagation at logic or higher abstraction levels. Addressing this correlation is essential for accurate soft error rate estimation, and more importantly, for the cross-layer error abstraction, e.g. from bit errors at logic level to word errors at Register-Transfer Level (RTL). Our proposed logic level analysis technique [19, 20] unifies the treatment of error-free signals and erroneous signals, and not only computes accurate output error probabilities when internal gates are impaired by soft errors, but also provides detailed quantification of the error correlations during the propagation process.

- **RTL error estimation leveraging word level error propagation properties**.

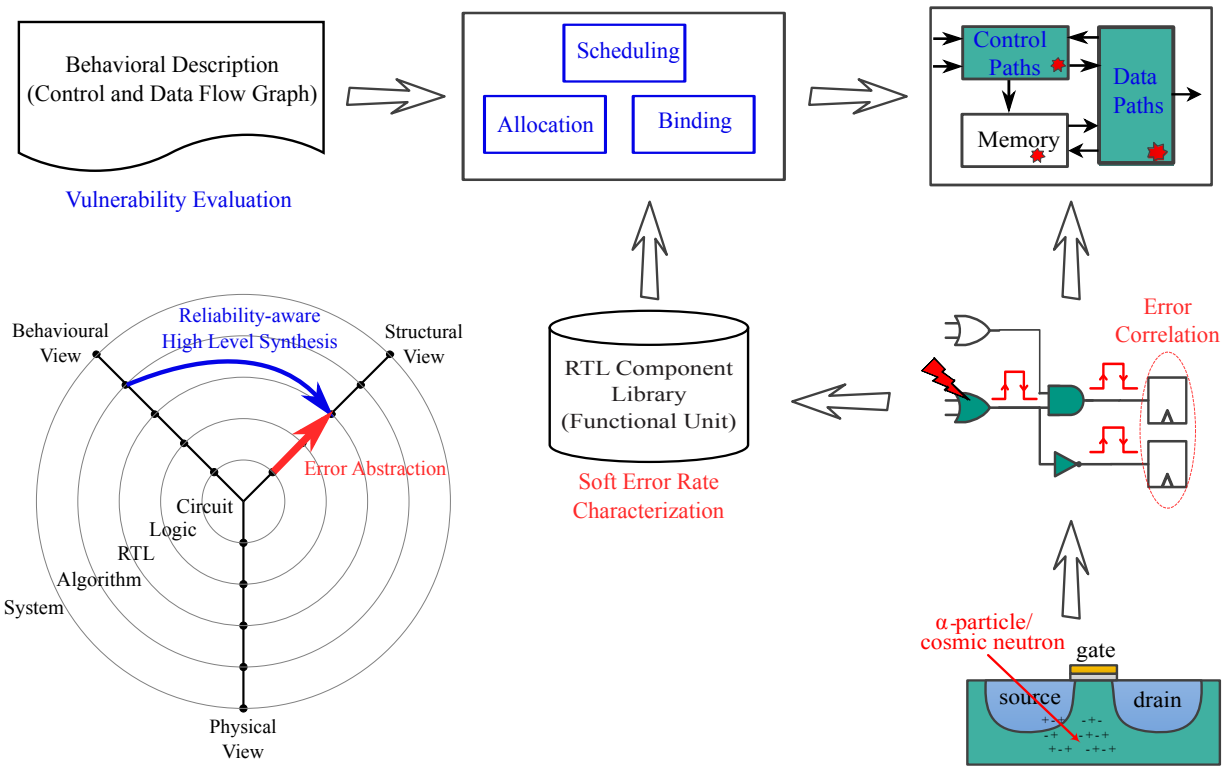


Figure 1.2: Contributions of this work visualized on the Y-chart

One drawback of the error estimation approaches at circuit/logic levels is that they are only applicable to flattened netlists, in which there is no design hierarchy and each signal is treated individually. Consequently, they lose the efficiency of error analysis and mitigation at higher level. To overcome this shortcoming, our proposed modeling work is extended from logic level to register transfer level [21–23]. It enables faster error estimation by utilizing the word-level abstraction efficiency at RTL, and moreover, quantifies the error contributions of different RTL components in a typical embedded processor by taking the workload dependency into consideration.

- **Behavioral variable and operation vulnerability analysis**.

  In addition to reliability characteristics of the RTL components, the behavioral variable and operation vulnerability, i.e. the probability of system failure given that component being erroneous, is another essential factor which influences the efficiency of selective protection and hardening. To have an accurate evaluation of these vulnerability values, with the behavioral design represented as a Control and Data Flow Graph (CDFG), we perform a comprehensive vulnerability evaluation with probabilistic model checking technique.

  These proposed cross-layer error estimation techniques take advantage of the abstraction efficiency and can successfully quantify the error contribution of each design component, therefore provide indispensable guideline for the following co-optimization of cost and reliability.

- **Reliability-aware resource binding during high-level synthesis**.

  During the High Level Synthesis (HLS) a behavioral description of the design at algorith-

mic level is translated to a structural RTL implementation. HLS stays in the early design phases, and provides the advantages of wide design space exploration and reduced design and verification efforts. In addition to conventional performance, power and area criteria, reliability-aware optimization during HLS is also investigated in our work [24, 25].

Motivated by the observation that for the behavioral designs, especially control intensive ones, the behavioral variables and operations manifest nonuniform error vulnerabilities, the co-optimization with other design objectives such as area and power is performed to improve the error mitigation efficiency by binding vulnerable behavioral constructs to few protected RTL components during HLS. The selective RTL protection guided by behavioral vulnerabilities can obtain the most cost-efficient soft error mitigation in the synthesized RTL implementation.

- **Reliability-aware scheduling during high-level synthesis**.

  Scheduling plays a critical role during the HLS process to explore the concurrent execution to improve the performance of the generated RTL design. In the scope of reliability-aware scheduling, we propose to redistribute the time within clock cycles according to the operation criticality, i.e. assign less time to execute non-vulnerable operations (using fast but less reliable RTL components) and more time to harden the vulnerable operations, so that the overall reliability of the generated RTL can be maximized with the same performance constraint. Our novel scheduling formulation takes into consideration not only the operation vulnerabilities at the behavioral level, but also the reliability and delay characteristics of RTL components.

1.2 Outline

The remainder of the thesis is divided into three main parts. In the first part, Chapter 2 contains the necessary background as well as the state of the art on soft error evaluation and mitigation. The second part (Chapter 3 and 4) focuses on the soft error evaluation using the proposed framework at various abstraction levels, from error correlation analysis at logic level to the vulnerability analysis at behavioral algorithmic level. The third part (Chapter 5 and 6) discusses how to make use of the evaluated reliability information to achieve cost-efficient error mitigation during the process of high level synthesis.

In particular, a brief introduction of each chapter is described as follows:

Chapter 2 contains the technical background on soft errors, which includes the origins of soft error occurrence, the error analysis and mitigation at different abstraction levels. In addition, the related work and the missing parts in the state of the art are also discussed.

In Chapter 3 a novel error estimation method is proposed to take into consideration the error correlations at logic level. The employed concept of error propagation function and super gate representation, which unify the treatment of error-free and erroneous signals, are described. It not only calculates both signal and error probabilities in one pass, but also takes the complex correlations among them into account. Two efficient heuristic algorithms are also introduced to improve the scalability of the proposed method.

Chapter 4 raises the error abstraction to register transfer and behavioral levels, and performs the error analysis by leveraging the word-level semantics. According to the different error

propagation properties, the RTL data paths are analyzed with analytical formulas of error propagation probability, while the control paths are modeled as state transition systems, and formal probabilistic model checking is adopted to quantify the soft error vulnerabilities of the registers in the control modules. In addition, the vulnerabilities of behavioral variables and operations are also investigated with similar model checking technique.

Chapter 5 proposes a new reliability-aware allocation and binding technique during high level synthesis. By adjusting the binding relations, i.e. mapping most vulnerable behavioral structures to few RTL resources, and only applying selective protection to those resources, cost-efficient reliability enhancement is achieved. The integer linear programming formulation of this reliability-aware optimization as well as a hardening-efficiency guided greedy algorithm to improve the scalability of the exact method is introduced.

Chapter 6 looks at the high level synthesis process from the timing perspective, i.e. the scheduling step is investigated for the potential of reliability enhancement. This chapter includes both the integer linear programming formulation and heuristic algorithm of the reliability-aware scheduling technique, which simultaneously considers the behavioral operation vulnerabilities and RTL reliability-cost trade-offs of the functional units.

Finally, Chapter 7 concludes the thesis and discusses the potential directions of future research.

2 Preliminaries and State of the Art

In the first part of this chapter, we will present the fundamental knowledge on soft errors, including the error origins, models, evaluation metrics and scaling trends. In the second part, the existing techniques to evaluate and mitigate soft errors at various abstraction levels will be discussed.

2.1 Soft Error Basics

Soft errors are radiation-induced transient hardware malfunctions due to the interactions between striking particles with high energy and semiconductor materials of the transistor. According to different types of circuits such as combinational gates, flip flops and memory cells, soft errors are modeled and analyzed in different ways. With the continuous technology scaling, the system SER is also increasing which exacerbates the error mitigation challenges in future.

2.1.1 Origins

For semiconductor devices, there exist various causes of transient faults such as supply voltage drop due to simultaneous switching current flows in the power and ground networks, and radiation from the external environments [26]. In this work we focus on the latter radiation-induced transient soft errors.

When a transistor is struck by an energetic particle, this collision can induce localized ions - a track of electron-hole pairs which may be collected by the source and drain areas of the transistor. Figure 2.1 gives a brief illustration of this striking and ionizing phenomenon. The recombination of these deposited charge will form a short current pulse, which may flip the logic value in a memory cell or at the output of a logic gate and introduce a soft error. The smallest amount of charge which can cause soft error is the so-called *critical charge* [3]. These errors are called *soft* because the transistor itself is not permanently damaged by the strike. After the system is reset, the hardware will again perform correct functionality.

For terrestrial applications, there are two main radiation sources which cause the soft errors [9]:

- Alpha particles emitted by radioactive impurities in the chip packaging materials;
- Neutrons generated by the interaction between the cosmic radiation and the atmosphere around the earth.

In the following, we will discuss the nature of these particles and the mechanisms that they introduce errors in transistors.

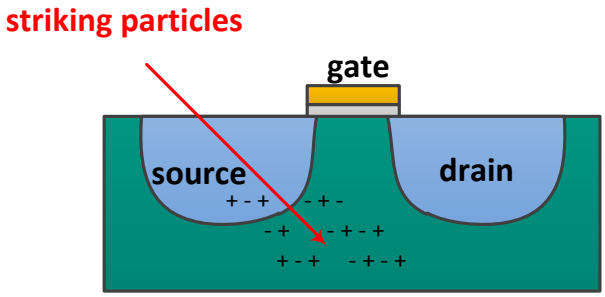


Figure 2.1: Particle striking and the induced ionization in a transistor

Alpha Particles

An alpha particle is composed of two protons and two neutrons, which are bound together and identical to one helium nucleus. The most common sources of alpha particles are from the radioactive nuclei, such as ^{238}U, ^{235}U, and ^{232}Th, in their decay process.

The first report on soft error, from Intel company in 1978, identified the alpha particles from the contaminated packing materials as the main error cause [10]. The alpha particles from the impurities in packaging materials typically have kinetic energies in the range from 4 to 9 MeV, which are lower than those of typical neutrons which affect Complementary Metal Oxide Semiconductor (CMOS) transistors. However, when these alpha particles pass through the semiconductor device, they lose the kinetic energy mainly through interactions with the transistor substrate, therefore deposit a dense track of charge and create electron-hole pairs.

The radioactive impurities, which emit alpha particles and cause soft errors, are mainly from the chip manufacturing and packaging processes, such as the semiconductor materials or the solder bumps. In particular, with the increasing use of flip-chip packaging toward 3-dimensional integrated circuits, the solder bumps have already moved very close to the active silicon devices. Therefore, even the low-energy alpha particles become capable to induce soft error [27].

There are different mitigation strategies to reduce the rate of alpha-particle-induced soft errors. The first one is to develop and manufacture the semiconductor chips using new materials with low radioactive compositions. The second one is to develop a barrier, either on the chip, directly under the solder bumps, or on the packaging above the thin-film wiring levels. This barrier can prevent the alpha particles due to external sources from affecting the internal product circuits [28]. However, both these approaches increase the processing complexity and manufacturing cost. It is very difficult, also very expensive, to completely eliminate alpha particles from the chip manufacturing and packaging process. Therefore, semiconductor chips typically need fault tolerant methods during the design to protect the chip against alpha particle strikes.

Cosmic Particles

Before 1980s radiation-induced reliability issue was only considered for the space applications. In space, the radiation conditions are much more severe than at the ground level due to

the *primary cosmic rays* and *secondary cosmic rays* [3]. The primary cosmic rays include galactic particles and solar particles, which bombard the outer atmosphere of the earth. While these particles collide with the atoms in the atmosphere, a shower of newly-created particles, including pions, muons and neutrons, consist of the secondary cosmic rays. The particles, which ultimately arrive at the earth surface, become the major threats of computing system at terrestrial level. Although protons and pions can also induce soft errors, and in particular, direct ionization of protons is promising to be a new concern in future technologies with smaller transistor size, currently the neutrons are considered as the most important source of radiation-induced soft errors [9].

The neutron is one of the elementary particles that make up an atom. It is an uncharged subatomic particle slightly heavier than that of the proton, which is positively charged. As neutron is neutral without electric charge, it does not interact with the charged electrons and can pass through the electronic clouds in atoms without being affected. Compared with electrons and other charged particles, neutrons are highly penetrating particles. That is to say, they can travel significantly large distance in the matter without obstruction. Therefore, they interact with atomic nuclei and become one origin of soft errors mainly via the production of secondary charged particles. This kind of mechanism is called *indirect ionization* [28].

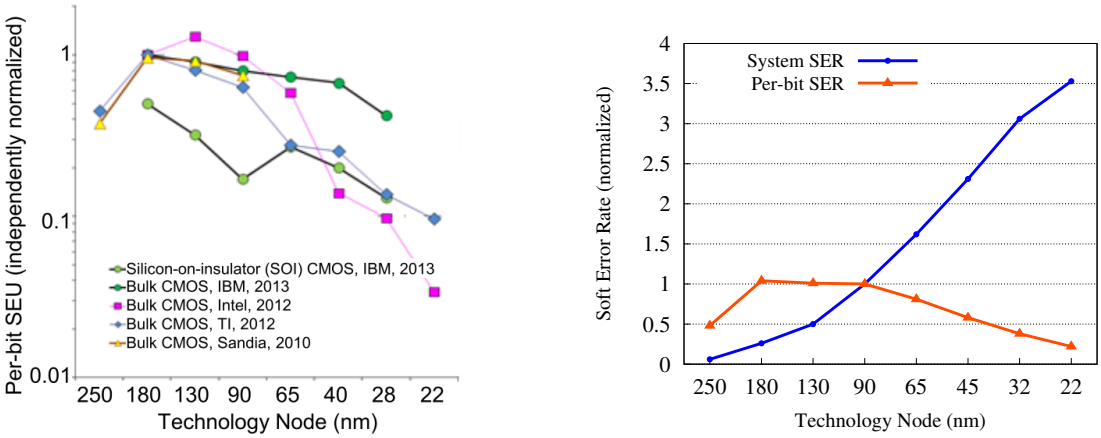
2.1.2 Models and Metrics

As a subset of Single Event Effects (SEE), soft error can be categorized into following models according to different circuit types [9, 29]:

- Single Bit Upset (SBU), which represents the radiation-induced bit-flip in a memory cell or a latch;
- Multi Bit Upset (MBU), when the striking particles have very high energy, they can cause the upset of two or more bits in the same word;
- Multi Cell Upset (MCU), when the striking event causes the upset of two or more memory cells or latches;
- Single Event Transient (SET), when the particle strikes a combinational logic gate, it can generate transient glitches in the gate output. If the induced SETs propagate in the gate network and finally are latched in downstream flip-flops or latches, they become soft errors;
- Single Event Latchup (SEL), when the particle strike triggers the parasitic bipolar transistors in CMOS technology. It creates abnormal large current and therefore induces a latch-up. For the occurrence of SEL, it is necessary to have a full chip power reset to remove this system malfunction. In some case SELs can also cause permanent damage to the device and introduce a hard error.

The term Single Event Upset (SEU) is generally a synonym for soft error, and in most case it is used to cover the two most common types of soft errors: SBU and MBU.

Regarding soft error metrics, usually the SER of a device is reported in Failure in Time (FIT), where 1 FIT means one failure per billion (10^9) operational hours, i.e. one failure per 144,077 years. The typical FIT rates of an electronic system is in the range from 100 to 100,000 FIT, which means approximate 1 soft error per year [9]. For soft error evaluation, the FIT metric is additive, which means the error rates (FIT) of a system is the summation of all the



(a) Per-bit SEU rates [31]

(b) Trends of bit and system SER predicted in [15]

Figure 2.2: Soft error trends for SRAMs

FIT rates of its consisting components [3].

$$FIT_{system} = \sum_{i=1}^{N} FIT_{component}^{i} \tag{2.1}$$

In the reliability engineering, Mean Time to Failure (MTTF) is another commonly used metric to express the system failure rates, which represents the mean time elapsed between two successive failures. As the soft errors are randomly introduced bit flips, it is reasonable to assume that the instantaneous SER of a system in a given time period is constant. Therefore, under the exponential failure law [30], the MTTF of a system and its soft error FIT rates are inversely related:

$$\text{MTTF in years} = \frac{10^9}{\text{FIT rate} \times 24 \text{ hours} \times 365 \text{ days}}$$

2.1.3 Scaling Trends

Following the famous Moore's law, the International Technology Roadmap for Semiconductors [32] predicts a drastic reduction of both the transistor supply voltage and physical gate size. According to this roadmap, in the next 10 years the number of transistors per system-on-chip (SoC) will increase by 12. In addition, the continuous increasing frequency of circuit operation also has significant impact on the evolution of SER. All these factors will exacerbate the soft error challenges in future technology nodes [28, 33].

Figure 2.2(a) shows the per-bit rate of SEU in SRAM. We can see that this rate has saturated and is even decreasing with technology downscaling. This is because not only the critical charge which can generate an SEU, but also the silicon volume from which induced charges can be collected, have decreased [9]. The latter trend dominates over the former one for the most advanced technology nodes. Nevertheless, the per-chip (i.e. system) SEU rate is still rapidly arising due to the continuously increasing integrity of transistors [15], as shown in 2.2(b). In addition, as the transistor size continues shrinking, the rate of single-event multiple

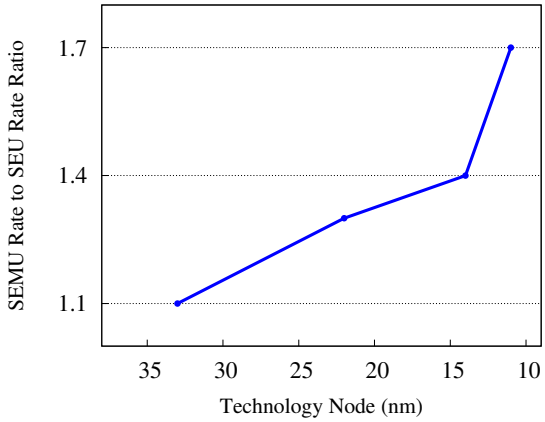


Figure 2.3: Single-event multiple upsets (SEMU) rate to SEU rate ratio for the SRAMs with Silicon-on-Insulator CMOS process [1]

upsets [34, 35], where a single particle strike can result in upsets at multiple circuit nodes, increases faster than that of SEUs, as shown in Figure 2.3.

Besides memory structures, due to the increasing operational frequency and decreased operating voltage, the latest experimental and radiation-testing results show that the rate of transient errors in sequential logic (i.e. latches and flip-flops) and combinational logic circuits is becoming comparable with that of memories [36–38]. Due to these exacerbating effects, accurate soft error analysis and the following efficient mitigation will become more challenging in the future technology nodes.

2.2 Soft Error Analysis and Mitigation

Driven by the continuously increasing market demands and technological advances, nowadays the complexity of computing system is growing rapidly with higher and higher integration density such as Multi-Processor System on Chip (MPSoC). The traditional design methods, in which the systems were designed directly starting from device or circuit level, are fast becoming infeasible in nano-scale era. To handle this complexity challenge, the common solution is to raise the abstraction level in the design process [39]. Well-defined abstraction levels and corresponding system models are particularly important to ensure accurate and efficient systematic design automation.

In this section we will firstly introduce the abstraction levels in a typical system design flows. Then we will turn to the specific soft error problem, and describe its analysis and mitigation solutions at both low and high abstraction levels in the state of the art. Particularly, the high level soft error analysis and mitigation techniques, which are the focus of this work, will be paid special attention.

2.2.1 Design Views and Abstraction Levels

In order to clearly explain the difference and connections among various abstraction levels, we adopt the well-known Gajski-Kuhn chart (i.e. Y-chart), which was developed in 1983 to depict

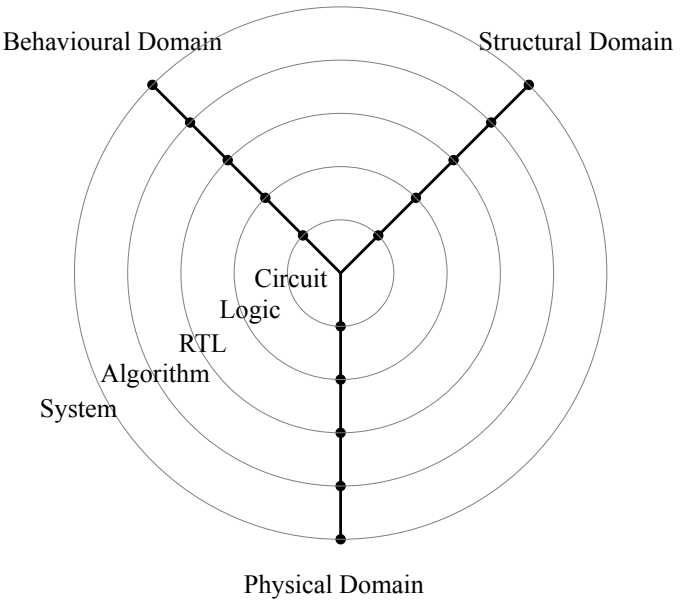


Figure 2.4: Gajski-Kuhn chart (Y-chart)

different stages involved in the system design flow [18].

According to this model, each design, no matter how complex, can be perceived in three domains (views), which represent different properties of this design. Specially, the Y-chart has three axes showing each domain:

- *Behavioural:* This domain describes the functional and temporal behaviors of the design, i.e. the outputs of the design can be represented in terms of its inputs over time;
- *Structural:* Here the subsystems which assemble the entire system are described, i.e. the structure of the system, different subsystems and their interconnections are described;
- *Physical:* For this domain dimensionality is added to the structure. It specifies the geometric properties of each component, i.e. information about the size (height and width), the shape and physical placement are included in this domain.

In the Y-chart a design can also be represented at different abstraction levels, i.e. the five concentric circles shown in Figure 2.4. Generally the outer circles are generalization (*abstraction*), while the inner ones are *refinements* of the same design. From the highest abstracted level to the most refined one, the five hierarchical levels are described as follows:

- *System Level:* The design specification is described as a set of subsystems and they loosely interact with each other (e.g. by exchanging messages). Typically block diagrams are used and the basic blocks in the structure domain are central processing unit, memory chip, etc.;
- *Algorithmic Level (behavioral level):* The specification describes the design functionality with algorithmic semantics including signals, loops, variables, assignments, etc.;
- *Register-Transfer Level:* With more detailed information, the design behavior are represented as values transfers between storage components, such as registers and register files, and functional units such as ALUs and multipliers. The data structures and data flows are also defined at this level;
- *Logic Level:* In the behavioral perspective the design is described as Boolean equations,

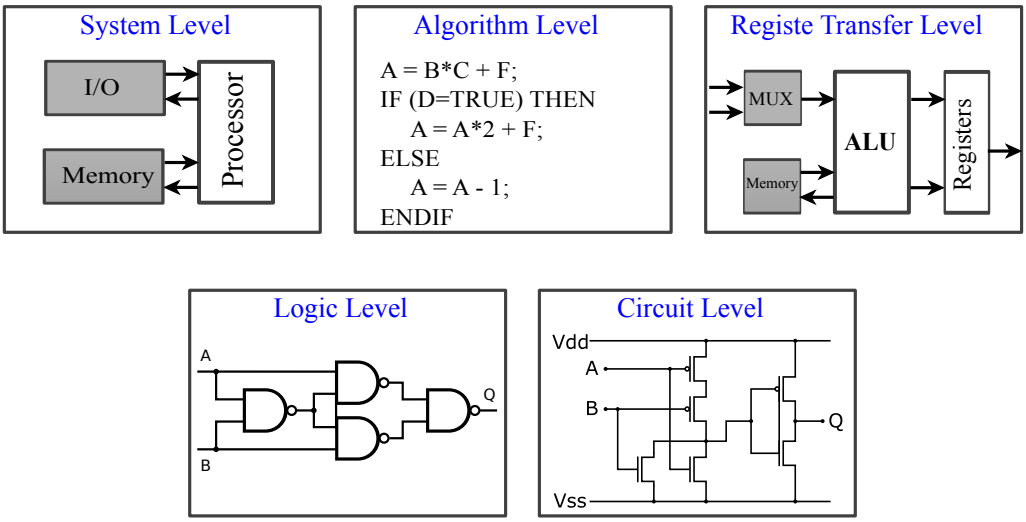


Figure 2.5: Illustration of the design at different abstraction levels

and in the structural view the basic constructing elements are logic gates, flip-flops and their interconnections;

- *Circuit Level:* At this level the circuit behaviors are represented by differential equations which define the mathematical relations between signal currents and voltage changes, etc. The structural elements consisting of the system are transistors, resistors and capacitors.

During the entire design flow, various design activities are involved, which can also be clearly demonstrated in the Y-chart, as shown in Figure 2.6. Among these activities, *abstraction* and *synthesis* are the main focus in our work on soft error analysis and mitigation. Therefore, we will describe them in more details.

Abstraction suppresses unnecessary details below specific granularity and is a commonly used approach to manage the complexity of computing systems. Due to the increasing complexity of current and future systems, high level design methodologies are more emphasized compared to low level techniques [39]. One reason for this trend is that high level abstractions are closer to the designer's usual way of reasoning about a complicated system. Hence, it is much easier to specify and verify various system functionalities with the help of abstractions, rather than modeling and analyzing a design starting from hundred thousands of concrete transistors. Another reason is that high level abstractions enable the system engineers to explore and verify the various design options in application domains without expert knowledge of low level technology engineering and chip manufacturing.

Synthesis is the transformation process of a design in the behavioral domain into a representation in the structural domain. The structural description of the same design after synthesis is typically formulated as interconnections of the abstract components. Automatic synthesis can significantly shorten the design cycle, and the associated design automation is a major contributor to reduce the well-known *productivity gap*, which is generated by the disparity between the rapid paces of increasing design complexity and the relatively slow rise of design productivity [32]. In addition, design optimization is usually performed in conjunction

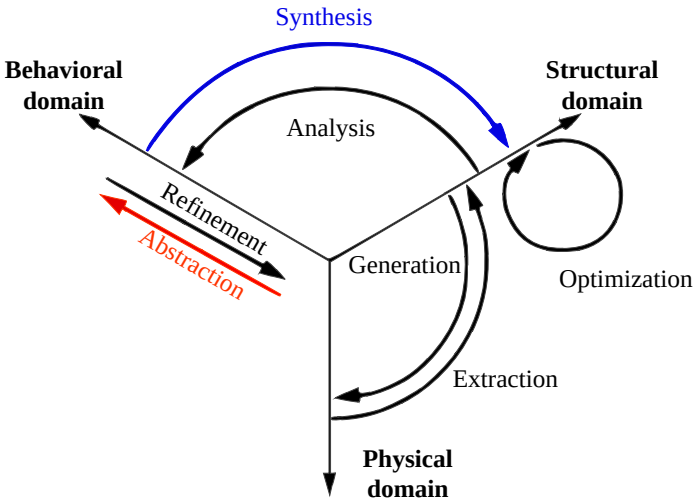


Figure 2.6: Y-chart and design activities [2]

with synthesis. The automatic synthesis, especially the high level synthesis staying at early design phase, adopts mathematical optimization algorithms to handle various design metrics of the system components, therefore they can explores the design trade-offs in a very efficient way [40].

Similar to the traditional design metrics like performance, area and power, for the specific SER metric its analysis and mitigation techniques can also be classified according to the abstraction levels. In this work we refer *low levels* as the circuit and logic abstraction levels in the Y-chart, while *high levels* represent the register transfer level and above.

2.2.2 Low Level Error Analysis and Mitigation

Soft errors are radiation-induced circuit malfunctions starting from the external particles striking the sensitive regions of semiconductor device. Therefore, the low level error analysis and mitigation techniques are always involved in charge collection, current pulse generation and transient pulse propagation in the circuit netlist.

Error Analysis

To obtain an accurate SER of a chip or its components, it is essential to have in-depth knowledge of the circuit elements (e.g. latches, gates, etc.) and their corresponding sensitivities to soft errors. Transient pulse generation and propagation at the gate output are the basis of SER evaluation. A commonly used pulse model is a double exponential current as described in [41, 42]:

$$I_{exp}(t) = \frac{Q_{tot}}{\tau_\alpha - \tau_\beta}(exp(-\frac{t}{\tau_\alpha}) - exp(-\frac{t}{\tau_\beta})) \tag{2.2}$$

where Q_{tot} is the total injected charge due to particle strike, τ_α is the charge collection time-constant of the junction, and τ_β represents the ion-track establishment time-constant. Both τ_α and τ_β are constant factors related to the technology process.

For older technologies, this double exponential current waveform can provide good ap-

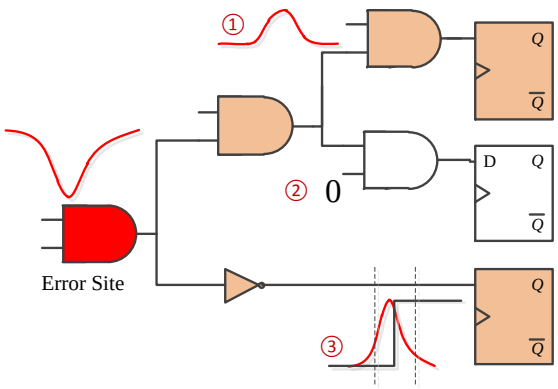


Figure 2.7: Three masking effects in the combinational logic

proximation of the charge collection process. However, for advanced technologies the shapes of current pulses are more complex, therefore, more accurate but also more time consuming approaches are introduced [43, 44]. They use either a device simulator working in full 3-dimensional model of the complete circuit (e.g. 6T SRAM cell), or mixed mode simulation with one transistor with a device simulator and the remaining transistors with a circuit simulator.

After current pulses generation modeling, how to analyze their propagation and masking behaviors in the gate-level network is another important issue. On the one hand, for sequential elements such as flip-flops, the timing masking efforts need to be carefully taken into account, when the SEU occurs too late in the clock period so that it cannot arrive at the input of downstream flip-flops within their latching windows [9]. On the other hand, when the current pulses are generated at the output of combinational gates, during their propagation process three masking mechanisms can take effects, which are illustrated in Figure 2.7 and described in the following [3]:

1. *Electrical masking.* After the current pulse is generated by particle strike, during its propagation alongside a chain of gates, typically the electrical strength of the pulse continues to attenuate. To be more specific, the rise and fall time of the pulse is increasing and its amplitude is decreasing. These changes are mainly due to the transistor switching delay and the turning off of a gate before the output pulse reaches the full amplitude, respectively. If the radiation-induced current pulses are completely attenuated before they reaches the downstream flip-flops, the soft error is said to be electrically masked. Either direct SPICE simulation or analytical models on the rise and fall time or the current waveform [45–48] can be employed to evaluate the electrical masking effect;
2. *Logic masking.* The generated current pulses arrive at the input of a gate, but its output value is completely determined by other controlling input. For example, for an AND gate with one input '0', the current pulse at another input will be masked and not cause an error. Typically for logic masking effect evaluation, either a logic-level simulator [49] with flipped inputs from '0' to '1' or vice versa, or analytical methods with probabilistic modeling of the input statistics [20, 50–52] can be used;
3. *Timing window masking.* If the current pulses are not electrically or logically masked, they can arrive at the inputs of downstream flip-flops. However, an edge-triggered flip-flop is only vulnerable to these propagated current pulses when they arrive within a small

timing window. Typically the size of this timing window is the sum of the setup and hold time of the flip-flop [9]. The current pulses outside this timing window can not be captured by the flip-flop. Assume the clock period is T_{clk} and the current pulse width at the input of flip-flop is PW, the timing window masking probability can be accurately estimated as $1 - PW/T_{clk}$ [9, 49, 53].

With these established models, the low level soft error estimation techniques can be generally categorized into statistical fault injection [54–56] and analytical approaches [8, 46, 47, 50, 57–60]. Fault injection techniques are based on Monte-Carlo simulations and to achieve a reasonable accuracy, they require long simulation time. Analytical probabilistic approaches have been developed and can be orders of magnitude faster than fault injection techniques. However, the accuracy and efficiency of analytical approaches rely heavily on the error and error propagation models. The circuit-level analytical techniques presented in [46, 47, 57–59] can handle the aforementioned three masking effects with detailed models and obtain accurate results for soft error rate estimation, but they do not have good scalability and are not suitable for error analysis at high level.

Error Mitigation

When the SER of the original circuit exceeds the user reliability specification, soft error mitigation schemes should be employed. The first category of low level mitigation techniques is manufacturing process improvement. For instance, the removal of Boron Phosphosilicate glass materials has significant contribution to reduce the soft errors [14]. In addition, the adoption of advanced Silicon-On-Insulator (SOI) process technology is another very efficient solution to reduce soft error rates, because unlike bulk CMOS, in SOI devices the collected electrical charge from particle strike is much less due to the much thinner silicon layer [28].

The second category of mitigation techniques is the design-based approach. Triple Modular Redundancy (TMR) is the most reliable solution for reducing the SER of a computing system. However, TMR-based mitigation solutions introduce excessive area and power overhead, hence are not applicable to most embedded system designs. To reduce the massive overhead, for sequential logic radiation-hardened storage cells (latches and flip-flops) such as Dual Interlock Storage Cells (DICE) [61], Built-in Soft Error Resilience (BISER) technique [38], Single Event Upset Tolerant [62] and Reinforcing Charge Collection [63] can be employed. They preserve the storage value even if the state of one of the internal nodes is altered by an ionizing particle strike, therefore reduce the intrinsic soft error rate.

We take the widely known DICE latch in Figure 2.8 as an example to explain the basic idea. The output node of each N and P transistor pair is the controlling input of another N and P transistor pair, i.e. “dual interlock”. When one of the output nodes is affected by transient current pulse due to particle strike, it can turn off the successively connected P and N transistors. Therefore, the turning off can isolate the impaired node and preserve the original state values in unaffected nodes. Once the generated transient pulse disappears, the preserved original value of the latch can be restored. According to the data from real radiation testing [33], with 80% area and 60% power overheads, the SER of the DICE-hardened flip-flop can be reduced by 10X compared with the unprotected one. In addition, a DICE latch does not increase the *d-to-q* time compared with the unhardened latch, therefore it is very attractive for the high speed design such as the processor pipeline [3].

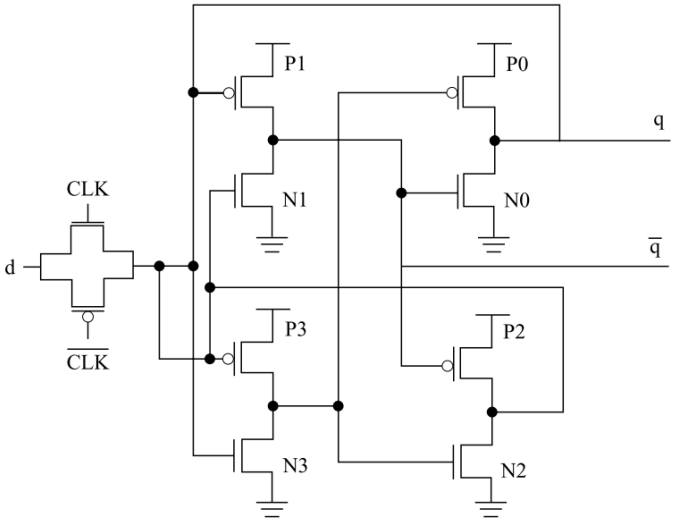


Figure 2.8: Schematic of a DICE latch [3]

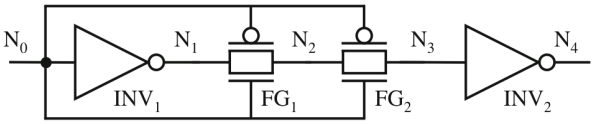


Figure 2.9: Transient filter schematic [4] with the Filter Gates (FG) driven by the input node N_0

For combinational gates, to reduce the SET sensitivity, gate resizing [64, 65] can be employed to increase either signal strength or the node capacitance, therefore raise the critical charge of the circuit node, i.e. reduce its soft error susceptibility. In addition, transient filters can also provide a considerable SER reduction by imposing small area and power overheads [4]. As logic gates have non-zero inertial delay, for the input current pulses which have smaller width than this delay, the gates can suppress the pulses from passing through. As shown in Figure 2.9, the basic idea of transient filters is to either completely remove the current pulse, or suppress its magnitude and duration to reduce the transient capturing probability of downstream flip-flops.

2.2.3 High Level Error Analysis and Mitigation

At high abstraction levels the detailed implementations of the design are transparent to the designer, therefore the soft errors are typically modeled as bit upset in the system states. The mitigation techniques are typically selective protection by exploring architectural and application level masking efforts, or software-based error detection and correction.

Error Analysis

Soft errors occur when particles strike the transistors and the collected charge is sufficient to change the logic state of this transistor. At RTL and above, this malfunction simply manifests as a bit flip in RTL registers, the architectural states, memories or behavioral variables. So the error model adopted at these levels are typically SBU or MBU [3].

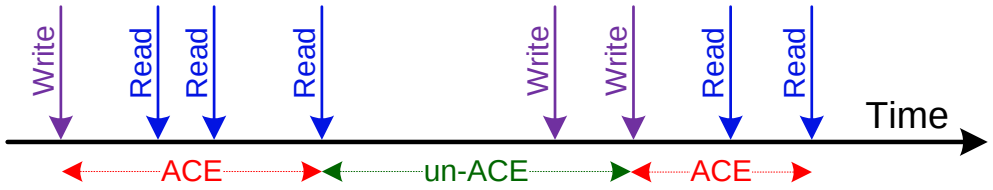


Figure 2.10: ACE and un-ACE intervals in the generic ACE analysis

After the occurrence of soft errors at low levels and during their propagation alongside the abstraction stacks, they can also be functionally masked at architecture [8] and application [66] levels. For the former architectural level masking, one example is that the soft errors, which occurred in branch predictor structures of a processor, only affect the system performance rather than the functional correctness. Regarding the latter application level masking, one example is that the errors propagated to the program variables, which are used in the procedure of random number generation, would be considered as benign, because these erroneous variables result only a different random sequence, but the system functionality is still correct. To obtain accurate error analysis, the functional masking effects at high abstraction levels also need to be carefully considered.

To achieve this goal, simulation-based fault injection is widely adopted to evaluate the system SER [67, 68]. The simulation-based fault injection has fair scalability, but it suffers from incomplete input space coverage and relatively low accuracy. Therefore, RTL formal methods, which feature exhaustive search of the design space and generation of the evaluation results with high confidence, are also widely researched [5, 6, 69]. However, the techniques in [6, 69] only consider qualitative error metrics (i.e. not quantitative probabilistic values), while the work in [5] cannot model the workload dependency during error evaluation.

For a bit flip generated by particle strike, from the application perspective whether it eventually influences the outcome of a program or not depends on how the bit flip propagates in the system and whether it is masked during this propagation process. To evaluate this masking effects at architectural level, the well known *Architectural Vulnerability Factor* (AVF) metric is introduced [70]. It is defined as the probability that a fault in the processor structure will result in a visible system error in the final output of a program. A bit, in which a fault will result in incorrect system execution, is seen to be necessary for Architecturally Correct Execution (ACE), and such kind of bits are termed ACE bits. All other bits are conservatively considered to be un-ACE bits. An individual bit may be ACE for a fraction of the overall execution cycles and un-ACE for the rest. Therefore, the AVF of a single bit can be defined as the fraction of cycles that the bit is ACE [8].

This generic ACE model is explained with respect to a simple example illustrated in Figure 2.10, which shows an example of serial accesses to a particular memory entry during the execution of a workload. In this model, all time intervals which lead to a write access are un-ACE as the write access overwrites the possible error. In contrast, an error in a time interval leading to a read access is always propagated to the memory output and has a chance to impair the final output of the program. Hence, such intervals are assumed to be ACE. The AVF of each memory entry is the fraction of time that it is in the ACE state, and the vulnerability of the entire memory is the average vulnerability of all memory entries.

Beside the generic ACE model, there are also more advanced techniques which exploit the architectural information to identify more non-vulnerable time intervals, and hence provide less pessimistic results. Such techniques leverage the specific behavior of each memory component such as data characteristics or access patterns to further reduce the ratio of ACE intervals to the entire execution time. For instance, there are dedicated techniques for data cache [71, 72], instruction cache [73], L2 cache [74], and register-file [75] which are less conservative than the generic model. Nevertheless, there is still a considerable gap between the failure probability obtained from fault injection and ACE analysis results (7X [76]).

Although the ACE approach has considerably short runtime which is almost equal to that of just one fault injection in a simulation environment, the significant overestimation limits its application on the soft error estimation. In addition, this technique is only suitable for the regular structures like memory units and register files, but not applicable to the irregular structures such as controllers with combinational and sequential elements [77].

Error Mitigation

For the high level soft error mitigation, there are both hardware and software solutions. In the hardware side, modular redundancy, as a general reliability enhancement scheme, can be applied at high abstraction levels as well. However, to avoid the introduction of massive overhead, the only viable solution is pre-evaluation of the vulnerability of individual component and then applying selective protection [6, 78]. For memory protection, the coding techniques [79, 80] nowadays become the popular low-cost solutions for the SER mitigation. The typical techniques, such as parity bit or ECC, can always be used for a large number of bits, e.g. memory words or blocks. Take the single error correct double error detect ECC code as an example, for each 64 bits data it only need 8 additional bits (around 12% overhead), which is much smaller than the typical 2X overhead of TMR.

As coding techniques are only applicable to regular memory structures, for the irregular controller and data processing blocks, another category of approaches for soft error mitigation are introduced. They reside in the scope of high level synthesis, which performs the automatic translation of a design from its behavioral description to a structural RTL implementation. In the recent work [81–86], soft error-induced reliability is taken into account in addition to the traditional metrics such as performance, area and power. These works explore the potential of reliability improvement on the resource allocation and binding process of high level synthesis. They leveraged the RTL library information to explore the system reliability enhancement, by adjusting the binding relations between behavioral operations and multiple implementations of the RTL functional units with different area and reliability metrics [81–86]. However, the important behavioral reliability information, which can capture the application level masking effects, is totally ignored in the previous methods. This can lead to inappropriate priorities of hardware protection and introduce unnecessary cost.

Software-based mitigation techniques are another attractive solution category which can either cooperate with hardware resilient measures, or just modify the software part to detect and correct errors by introducing redundancy without changing the underlying hardware. This is known as *software-implemented hardware fault tolerance* [87, 88]. Such software can not only perform the original functionalities according to the user's specification, but also support special mechanisms to detect, signal and even correct the possible occurrence of both permanent

and transient hardware errors. These techniques include computation duplication [89], executable assertions [90] to address the faults affecting the data flow, and control flow checking using assertions [91] or signatures [92].

2.2.4 Summary of State of the Art

The previous soft error estimation and mitigation techniques can be generally categorized according to their abstraction levels. At low abstraction level soft error estimation is performed either by statistical fault injection or analytical approaches. Fault injection techniques are based on Monte-Carlo simulations, therefore to achieve a reasonable accuracy, they require huge simulation time. Analytical probabilistic approaches can be orders of magnitude faster than fault injection techniques. However, their accuracy and efficiency rely heavily on the error and error propagation models. The circuit-level analytical techniques can handle logical, electrical and timing window masking, but they do not have good scalability and are not able to capture the masking effects at high abstraction levels (e.g. architectural and application level). Moreover, most of existing solutions are based on single error assumption and do not consider MBUs or multiple correlated errors to next abstraction levels.

At RTL, the current error estimation techniques can only provide qualitative error metrics rather than quantitative values, or cannot capture the application dependency during error evaluation. At architecture level, most current techniques rely on the ACE analysis, which is suitable for the regular structures like memory units and register files, but not applicable to the irregular structures such as controllers with sequential elements.

To enhance the reliability of future embedded system designs, while ECC can be employed to efficiently protect the regular memory blocks, the sequential elements (flip-flops) and combinational logic gates become the dominant contributors to the overall system SER. Traditional reliability enhancing techniques such as duplication or triple module redundancy, as well as circuit level radiation-hardening cells can introduce high performance, area, and power overhead, making them inapplicable to current and future reliability challenges of embedded systems. Therefore, cross-layer soft error resilient techniques based on selective protection, which take into consideration not only low level reliability information but also the application level masking effects, are still missing in the literature.

In the following chapters we will discuss in details how our research work advances the state of the art by novel cross-layer error evaluation and mitigation techniques.

3 Error Correlation Analysis at Logic Level

Single or multiple event transients at low levels can result in multiple correlated bit flips at logic or higher abstraction levels. Addressing this correlation is essential for accurate soft error rate estimation, and more importantly, for the cross-layer error abstraction, e.g. from bit errors at logic level to word errors at register-transfer level. This chapter introduces a novel error estimation method to take into consideration both signal and error correlations. The proposed method not only reports accurate error probabilities when internal gates are impaired by soft errors, but also gives quantification of the error correlations in their propagation process. This feature enables our method to be a versatile technique to be used in high-level error estimation.

3.1 Introduction

In digital circuits, a single event transient at one circuit node can propagate to multiple outputs, and be captured by multiple flip-flops in the same clock cycle. The way this single event transient is seen at higher abstraction levels could typically manifest as *multiple correlated* errors (bit flips). This correlation is particularly important for high level error modeling of the commonly used multi-bit data/control words, intellectual property macro interfaces and so on. Assuming independence among these bits can severely reduce the accuracy of soft error modeling at higher abstraction levels, and in turn the efficiency of the mitigation techniques.

However, currently at register-transfer or higher (e.g. architectural) levels, single bit-flip or multiple *independent* bit-flips are assumed and these errors are randomly injected for simulation-based techniques [55, 93]. The same assumption is used for analytical approaches as well [8, 60]. Although circuit and logic level techniques consider detailed erroneous glitches for accurate soft error rate estimation [46, 47, 57–59], they do not quantify correlations among different erroneous signals.

To close this gap between low level error estimation and high level error abstraction, we proposes a novel method to explicitly consider and report quantified error correlations in soft error analysis. It is based on the concept of *error propagation function* and a new *super gate* representation to unify the treatment of error-free and erroneous signals. It not only calculates both signal and error probabilities in one pass, but also takes the complex correlations among them into account. The correlation coefficient method originating from signal probability calculation is adjusted to obtain error probabilities and correlations of primary outputs due to particle strike at internal nodes.

In addition, two efficient heuristic algorithms are introduced to further improve the scalability of our method:

- *Dynamic blocking of error propagation* dynamically sets gates with very low error probabilities as error-free to speedup the calculation with negligible accuracy loss.
- *Limited depth correlation analysis* reveals the opportunity to consider only highly corre-

lated signal pairs to significantly improve the scalability.

Compared with the preliminary implementation in [94], these heuristics make our framework reach three orders of magnitude speedup, while still maintaining satisfactory accuracy. In addition, in this chapter we also present a detailed case study of OpenRISC ALU to illustrate the necessity of error correlation analysis and the applicability of our proposed methodology. The experimental results show that our method is five orders of magnitude faster than Monte-Carlo simulation while the average inaccuracy[1] of error probability estimation is only 0.02.

The organization of the rest of this chapter is as follows. Section 3.2 reviews related work on soft error and circuit reliability estimation. Section 3.3 introduces our adopted correlation model. Sections 3.4 discusses different procedures to calculate correlated error probabilities based on the proposed super gate representation and Section 3.5 discusses the extensions of our method. Section 3.6 describes the experimental results. Finally, this chapter is concluded in Section 3.7.

3.2 Related Work

Soft error estimation techniques can be generally categorized into statistical fault injection [54–56] and analytical approaches [8, 46, 47, 57–60]. Fault injection techniques are based on Monte-Carlo simulations and to achieve a reasonable accuracy, they require long simulation time. To overcome the scalability of fault injection, analytical probabilistic approaches have been developed and can be orders of magnitude faster than fault injection techniques. However, the accuracy and efficiency of analytical approaches rely heavily on the error and error propagation models. The circuit-level techniques presented in [46, 47, 57–59] handle three masking effects, namely logical, electrical and timing masking, with detailed models and obtain accurate results for soft error rate estimation, but they do not have good scalability and are not suitable for error analysis at high level.

Additionally, in the scope of correlation modeling for logic errors (bit-flips), the technique proposed in [58] uses random vector simulation to estimate the probability of logical masking, which does not give error correlations and furthermore, requires high simulation time for large circuits. In the techniques [46, 47, 57, 59], only the error probabilities of primary outputs are obtained without providing their correlations, which, however, are indispensable for lifting the abstraction level of soft error analysis. The error propagation probability technique [95] has better scalability with respect to runtime, but it addresses only the reconvergent effects of error propagation using 4-value logic. The error-free signals are assumed to be uncorrelated, even for the inputs of the reconvergent gate. The inaccuracy due to this simplification impairs its runtime advantage and again, it cannot report the error correlations at circuit outputs.

With bit-flip model, soft error probability estimation at logic level can also be handled by circuit reliability estimation techniques [50, 52, 96–99], where each gate is assumed to be unreliable and has a specified failure probability. Regarding the modeling of error correlations, probabilistic transfer matrix approach [50] employs algebraic decision diagrams to facilitate matrix operations and capture correlations among internal signals, but it suffers from massive matrix storage and high runtime. Bayesian network approach [96] uses conditional probability

[1] As Monte-Carlo simulation can not obtain the *exact* results either, the *inaccuracy* here actually means the deviation of our estimated error probabilities from the results reported by Monte-Carlo simulation.

table for each gate to capture all internal signal correlations, however, manipulating large circuits using Bayesian network potentially need very high runtime.

One-pass reliability method [52] uses correlation coefficients to model the correlations of two bit-flip events, 0→1 and 1→0, on different wires. Four correlation coefficients are necessary for each pair of wires and up to 16 correlation coefficients if higher accuracy is required. Trigonometry-based probability calculation [98] uses trigonometric functions to manipulate signal and error probabilities, and obtains accurate results for low error probabilities ($<10^{-4}$), but it is not appropriate for large error probabilities. Boolean Difference-based Error Calculator [99] adopts mathematical differential equations to model error propagation, and several levels of logic from the fanout node are collapsed into one level to handle reconvergent effects, however, its scalability is questionable for larger correlated regions. The hybrid approach [97] combines exact analysis with probabilistic measures to estimate reliability, and establishes algebraic decision diagrams for each reconvergent region, which limits its application to large circuits.

To sum up, in the scope of error correlation with bit-flip model, the related work mentioned above suffers from at least one of the following disadvantages:

- Only the error probabilities of primary outputs are obtained, therefore it cannot report quantified error correlations.
- Due to the intrinsic scalability issue its applicability is always limited to small or medium size circuits.

Therefore, in this chapter we propose a new technique to overcome these two disadvantages, aiming at quantification of error correlations to support hierarchical soft error analysis and scalability enhancement to handle large size circuits.

3.3 Correlation Model

In typical digital circuits, there are two kinds of correlations: *temporal* and *spatial* correlations [100]. Temporal correlation is related to the historical trends of bit streams, which is always considered in switching activity calculation and it is beyond the scope of this work on error propagation estimation in combinational circuits. Spatial correlation, which describes the dependencies between signals at different locations in the netlist, originates from two main sources:

- *Structural dependence*: It is due to the reconvergent fanout, where two or more signals originate from the same gate, propagate on different paths and converge again to the inputs of another gate.
- *Primary input dependence*: It refers to the spatial correlations among primary input signals, which result from input vectors with workload dependencies.

The most familiar measure of dependence between two variables is the *Pearson product-moment Correlation Coefficient (PCC)*. It is obtained by dividing the covariance of the two variables with the product of their standard deviations [101]:

$$\rho_{X,Y} = \mathrm{corr}(X,Y) = \frac{\mathrm{cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y} \tag{3.1}$$

where X, Y are random variables, μ_X, μ_Y are the mean values and σ_X, σ_Y are the standard deviations. If X are Bernoulli variables (e.g. the logic signal with binary values), we know that

$$E(X) = P(X), \quad \sigma_X = \sqrt{P(X)(1 - P(X))} \tag{3.2}$$

where $P(X)$ is the probability of X, therefore the PCC of two Bernoulli variables can be expressed as:

$$\begin{aligned} \rho_{X,Y} &= \frac{E[(X - P(X))(Y - P(Y))]}{\sqrt{P(X)(1 - P(X))P(Y)(1 - P(Y))}} \\ &= \frac{P(XY) - P(X)P(Y)}{\sqrt{P(X)(1 - P(X))P(Y)(1 - P(Y))}} \end{aligned} \tag{3.3}$$

The value of the $\rho_{X,Y}$ ranges from -1 to 1. A value of 0 implies that there is no linear relationship between the two variables, while the value of -1(1) implies they are perfectly negative(positive) correlated, i.e. whenever one variable has a high/low value, the other has a low/high(high/low)value.

3.3.1 Correlation Coefficient Method

PCC can clearly indicate the degree of correlation between two variables, but it is too complicated to be used for the correlation modeling in combinational network. To make it easier, *Correlation Coefficient Method* (CCM) [102] is adopted in our work for two main reasons. First, it provides accurate results, while has better scalability than the approaches in [50, 96]. Second, CCM can be easily extended to address the primary input dependencies as well [100].

Algorithm

Based on the topologically levelized circuit netlist, CCM propagates both signal probabilities and signal correlations level by level from primary inputs to primary outputs. It considers the correlations between any two signals in each level, therefore can handle both structural dependencies and primary input dependencies effectively.

In CCM with the notation of signal probability $p(i) \equiv P(i = 1)$, the *Correlation Coefficient (CC)* of signals i, j is defined as

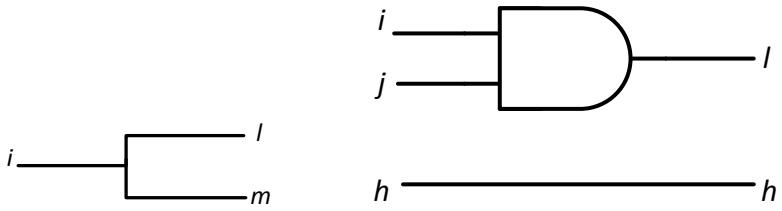
$$C_{i,j} = C_{j,i} = \frac{p(ij)}{p(i)p(j)} = \frac{p(i|j)}{p(i)} = \frac{p(j|i)}{p(j)} \tag{3.4}$$

where $p(ij)$ is the joint probability $P(i = 1, j = 1)$, $p(i|j)$ is the conditional probability $P(i = 1|j = 1)$, therefore $C_{i,j}$ is in the range $[0, +\infty)$.

Compared with Equation (3.3), it is clear that $C_{i,j}$ and $\rho_{i,j}$ can be derived from each other. For the special case that two signals are uncorrelated,

$$p(ij) = p(i)p(j) \rightarrow C_{i,j} = 1,\ \rho_{i,j} = 0$$

In this work, PCC is only used to indicate the degree of correlation between signals, due to its normalized value range $[-1, 1]$, while CC is used mainly for correlation propagation in the circuits.



(a) Fanout node (b) *AND* gate and wire

Figure 3.1: Typical structures for correlation calculation

Take an *AND* gate $l = ij$ as example, given p_i, p_j and $C_{i,j}$ we can exactly calculate the signal probability of the output l using the following formula:

$$p(l) = p(i)p(j)C_{i,j}, \quad 0 \leq C_{i,j} \leq \frac{1}{p(i)p(j)} \tag{3.5}$$

In addition, based on several basic propagation rules, the correlation coefficients between different signals can be analytically computed for all structural cases in the combinational network. Two typical cases are illustrated in Figure 3.1 and the corresponding correlation formulas are $C_{l,m} = 1/p(i)$ and $C_{l,h} = C_{i,h}C_{j,h}$, respectively. Note that in CCM for a gate with fanout > 1, each fanout is treated as a separate wire for the correlation calculation.

Accuracy Issue

In Figure 3.1(b) it is assumed that

$$C_{ij,h} \approx C_{i,h}C_{j,h} \tag{3.6}$$

thus the dependencies of two signals to a third one is neglected. Hence the second and higher order correlations among multiple signals are not taken into account in CCM. The signal probability estimation in [102] and switching activity estimation in [100] show this first-order approximation can provide accurate results in practice. However, neglecting high order correlations may cause the gate output probability to be smaller than 0.0 or larger than 1.0. Therefore, Inequality (3.5) is used to limit $C_{i,j}$ of the *AND* gate to avoid probability overflow. For other logic gates, similar inequalities can be derived for the purpose of probability bounding.

Actually, we find that the upper bound of correlation coefficient in Inequality (3.5) is rather loose. Revisiting the definition of correlation coefficient in Equation (3.4), we have:

$$\begin{aligned} C_{i,j} = \tfrac{p(i|j)}{p(i)} = \tfrac{p(j|i)}{p(j)}, \quad & p(i|j) \leq 1 \text{ and } p(j|i) \leq 1 \\ \Longrightarrow \; & C_{i,j} \leq min\{\tfrac{1}{p(i)}, \tfrac{1}{p(j)}\} \end{aligned} \tag{3.7}$$

This new inequality gives a tighter upper bound, therefore provides better error bounding in the correlation propagation, especially for the signals with high order correlation.

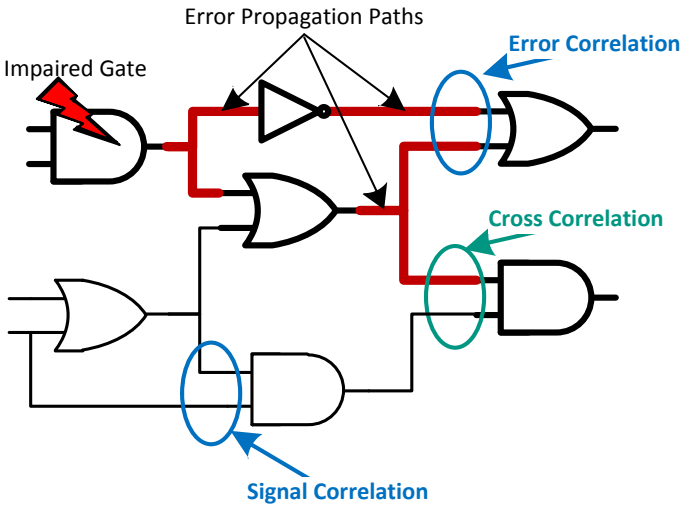


Figure 3.2: Different types of correlation

Complexity Issue

It is shown in [102] that the complexity of CCM is linear in the number of topological levels L and pseudo-quadratic in the number of gates per level N_L, which could be expressed as

$$\text{Complexity of CCM} \leq \sum_{L} N_L(N_L - 1)/2 \tag{3.8}$$

Please note that the right side of this inequality is the worst case, because to get output probabilities CCM only needs to calculate the correlation coefficients for these signals that are dependent from each other.

3.3.2 Error Correlation

In the scope of soft error modeling, the irregular structures in combinational logic make the error correlation modeling more complicated than that in the regular structures such as memory arrays.

After error occurs and propagates along multiple reconvergent paths, there exist various types of correlations, as shown in Figure 3.2. Not only the error-free signals, whose probabilities determine the logical masking effects, but also the erroneous signals, are possible to be correlated (referred as *Signal Correlation* and *Error Correlation*, respectively). To make things worse, even the signal probabilities of error-free signals and error probabilities of erroneous signals are not independent (referred as *Cross Correlation*). It is indispensable to consider all these correlations to achieve good accuracy for error estimation.

3.4 Proposed Error Estimation Methodology

Soft errors are modeled as bit flips in our work. For the sake of simplicity, we present our approach for the circuit mapped into a network of two-input elementary gates. Nevertheless, the presented approach can readily be applied to a network of arbitrary logic cells.

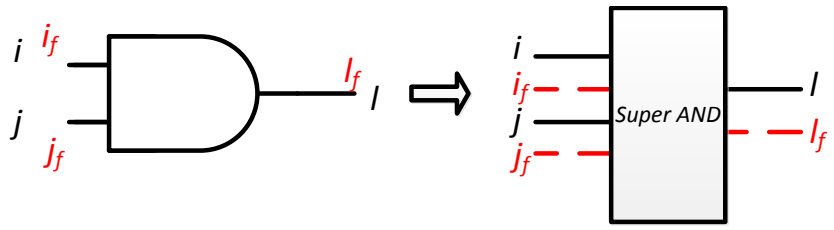


Figure 3.3: Super gate representation

The proposed methodology for error probabilities and correlations estimation is divided into three main parts. First, error propagation in the network of elementary gates is efficiently modeled. Second, combining this error propagation model with CCM enables us to estimate both error probabilities and correlations. Third, due to the intrinsic complexity issue of CCM, scalability enhancement techniques are explored to make our method applicable to large size circuits.

3.4.1 Error Propagation Model

To illustrate the basic idea, we take the Boolean *AND* function $l = ij$ as the running example. Totally there are four error-free input combinations. For $ij = 01$, only when i is erroneous ($0 \rightarrow 1$) and j is error-free, the output l is erroneous ($0 \rightarrow 1$). The other three combinations could be analyzed similarly.

If notation x_f is used to indicate whether the signal x is erroneous, i.e. $x_f = 1$ means a bit-flip occurs on signal x, caused by either particle strike or error propagation, the four cases for the *AND* gate can be combined and expressed with a single Boolean function, called *Error Propagation Function (EPF)*:

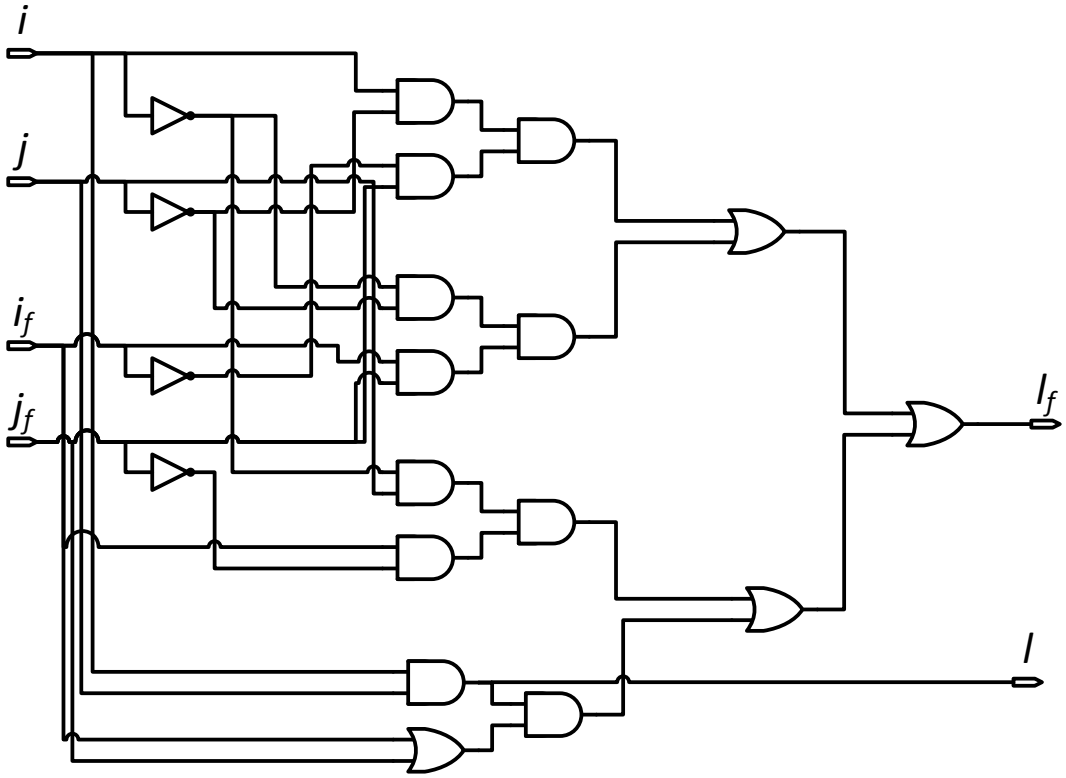
$$l_f = i\bar{j}\bar{i}_f j_f + \bar{i}\bar{j} i_f j_f + \bar{i} j i_f \bar{j}_f + ij(i_f + j_f) \tag{3.9}$$

here i_f, j_f, l_f are just virtual error signals to facilitate the modeling of error propagation, and do not exist in real circuits. Although the virtual error signals are treated as normal ones, their property interpretations are different:

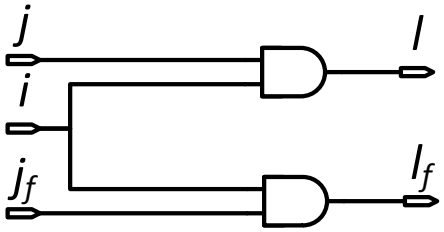
- *Logic value*: '1' is interpreted as bit-flip occurring in the corresponding real signal and '0' means error-free;
- *Signal Probability (SP)*: interpreted as the error probability of the corresponding real signal.

If the error-free function $l = ij$ is combined with EPF, a four-input, two-output *super gate* can be constructed, as illustrated in Figure 3.3. This representation has several important features:

- *It simplifies the propagation and correlation modeling of the bit-flips in combinational network*: it does not differentiate two kinds of bit flips: $0 \rightarrow 1$ and $1 \rightarrow 0$ as in [52]. To model all three types of correlations between two wires in Figure 3.2, our method needs only $\binom{4}{2} = 6$ rather than 16 correlation coefficients in [52].
- *The super gate representation is independent of any specific algorithm*: it is just an additional Boolean function and could be analyzed using the well-known methods in the



(a) Super *AND*



(b) Semi-super *AND*

Figure 3.4: Super and semi-super AND gate-level implementation

areas of signal probability [102], switching activity [100], power estimation [103], etc.

- *This concept can be easily extended to more complicated error modeling*: it can be adjusted for modeling single bit-flip, multiple bit-flips, transient errors, permanent errors, etc.

This modeling technique doubles the number of signals needed to be considered in the analysis. Recalling the runtime Inequality (3.8) and the intuitive gate-level implementation of super *AND* in Figure 3.4(a), it is clear that the super-gate implementation increases not only the circuit level L, but also the number of gates N_L at each level, therefore introducing high runtime overhead. This additional complexity is unavoidable to model propagation of bit-flip errors and their complicated correlations. Nevertheless, this complexity problem can be alleviated by exploration of the logic properties of EPF and limited correlation heuristic, as discussed later in this section.

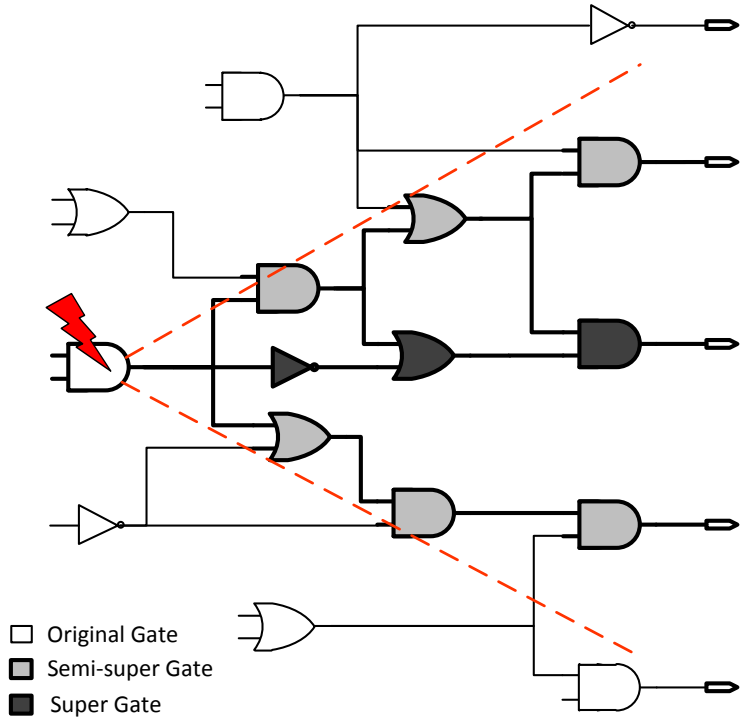


Figure 3.5: Error propagation paths and error cone

3.4.2 Error Cone Extraction

Due to the complexity issue of CCM in Section 3.3.1, the straightforward approach to uniformly replace each gate in the original circuit with its corresponding super gate is not feasible. Actually, in the scope of soft error only the gates in the fanout cone of the error site (i.e. *error cone*) will contribute to the error probabilities of primary outputs, as shown in Figure 3.5.

Moreover, for the gates at the boundary of the error cone, only one of its two inputs can be erroneous. This observation can be explored to efficiently reduce the complexity of super gate implementation in Figure 3.4. Recalling the EPF in Equation (3.9), and assuming the input i is error-free, i.e. $i_f = 0$, this EPF can be simplified as following:

$$l_f = i\bar{j}j_f + ijj_f = ij_f \tag{3.10}$$

In this way, the complex super gate collapses to only two basic gates, called *semi-super gate*: one for the error-free function, the other for the error propagation. Obviously, on the error propagation paths, the more gates collapsing into semi-super gate, the more benefit we get from this simplification, as shown in Figure 3.5.

3.4.3 Super-gate Correlation Formulas

To avoid replacing super/semi-super gates with their complex gate-level implementation and the corresponding netlist transformation efforts, it is necessary to derive error probability and correlation propagation formulas for super, semi-super and original gate pairs [94].

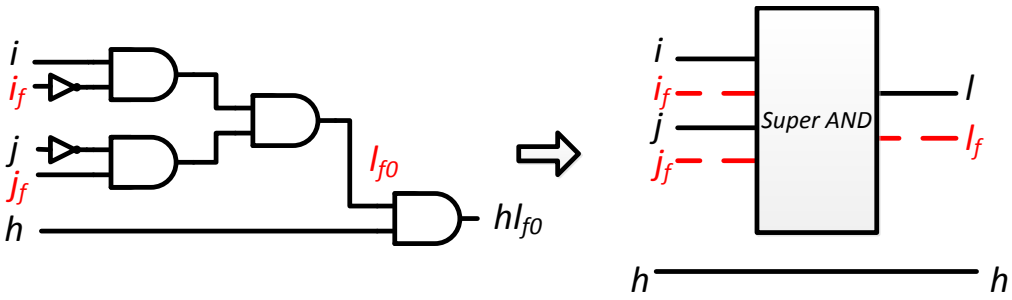


Figure 3.6: Super-AND and wire correlation

The EPF of *AND* gate in Equation (3.9) can be rewritten as

$$\begin{aligned} l &= ij \\ l_f &= \underbrace{i\bar{j}\bar{i}_f j_f}_{l_{f_0}} + \underbrace{\bar{i}\bar{j}i_f j_f}_{l_{f_1}} + \underbrace{\bar{i}ji_f\bar{j}_f}_{l_{f_2}} + \underbrace{ij(i_f + j_f)}_{l_{f_3}} \end{aligned} \tag{3.11}$$

One very important property of this function is that the four terms $l_{f_0}, l_{f_1}, l_{f_2}, l_{f_3}$ are *mutually exclusive*, which means

$$\begin{aligned} p(l_{f_i} l_{f_j}) &= 0, \quad i, j = 0, 1, 2, 3 \text{ and } i \neq j \\ p(l_f) &= p(l_{f_0}) + p(l_{f_1}) + p(l_{f_2}) + p(l_{f_3}) \end{aligned} \tag{3.12}$$

This logical property can be explored to simplify and accelerate the calculation of error probabilities, cross correlations and error correlations.

For the calculation of probability $p(l_f)$, each of the four terms could be calculated using basic gate rules introduced in Section 3.3.1, as every term contains only four variables. For the correlation coefficient C_{l,l_f}, instead of using the original CCM for gate-level implementation of EPF, with the Boolean Equations (3.11) we turn to correlation coefficient definition in Equation (3.4) to derive this value:

$$\begin{aligned} p(ll_{f_i}) &= 0, \quad i = 0, 1, 2 \\ C_{l,l_f} = \frac{p(ll_f)}{p(l)p(l_f)} &= \frac{p[l(l_{f_0} + l_{f_1} + l_{f_2} + l_{f_3})]}{p(l)p(l_f)} \\ &= \frac{p(ll_{f_0}) + p(ll_{f_1}) + p(ll_{f_2}) + p(ll_{f_3})}{p(l)p(l_f)} \\ &= \frac{p(ll_{f_3})}{p(l)p(l_f)} = \frac{p(l_{f_3})}{p(l)p(l_f)} \end{aligned} \tag{3.13}$$

where $p(l_{f_3})$ and $p(l_f)$ can be obtained from Equation (3.12) and $p(l)$ is the signal probability from error-free CCM.

To derive the general formulas for typical structures, e.g. Super-*AND*/wire pair in Figure 3.6, a similar approach can be applied to avoid unnecessary correlation computation efforts:

$$\begin{aligned} C_{h,l_f} = \frac{p(hl_f)}{p(h)p(l_f)} &= \frac{p[h(l_{f_0} + l_{f_1} + l_{f_2} + l_{f_3})]}{p(h)p(l_f)} \\ &= \frac{p(hl_{f_0}) + p(hl_{f_1}) + p(hl_{f_2}) + p(hl_{f_3})}{p(h)p(l_f)} \end{aligned}$$

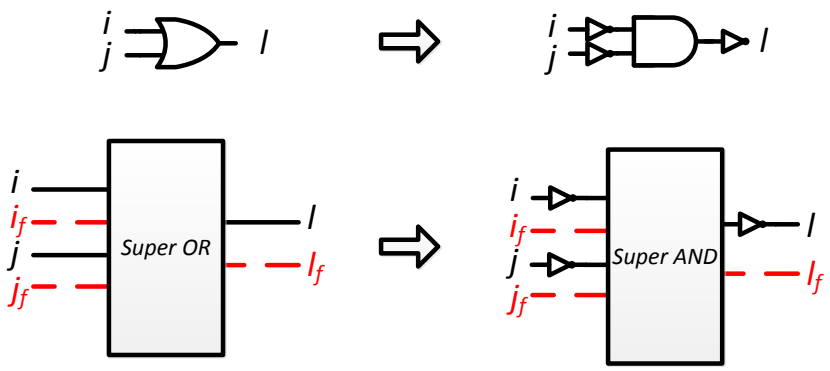


Figure 3.7: Super-AND equivalent of Super-OR gate

Based on the mutually exclusive property, the complex EPF is broken up into four small parts, each of which can be calculated quickly.

Using De Morgan's law, and since *INV* gate does not logically mask any error, the above Super-*AND* correlation rules can be extended to Super-*OR* with minor modifications as shown in Figure 3.7. Furthermore, for EPFs of the simplified semi-super gate, the derivation of corresponding formulas is straightforward and omitted here for brevity.

3.4.4 Dynamic Blocking of Error Propagation

Our focus is probabilistic analysis of the soft error propagation in combinational networks, and the area having erroneous signals is always limited to the error cone. In addition to the techniques introduced in Section 3.4.2 and 3.4.3, *dynamic blocking of error propagation* can also be employed to speedup the calculation.

This idea comes from the observation that due to logical masking, the longer the error propagation path is, the smaller the error probability could be. Therefore, when the calculated output error probability of one gate is smaller than predefined threshold value, this gate could be assumed to be error-free, i.e. its type changes from super gate or semi-super gate to original gate, its error probability is reset to constant zero and the correlations between its corresponding virtual error signal to all other signals are eliminated. The choice of this predefined threshold is a trade-off between scalability and accuracy, and in our implementation this value is set to 10^{-4}. Note that this gate type change may cause the type of its successor chain to be dynamically adjusted, e.g. from super gate to semi-super gate or semi-super to original gate. Therefore, this dynamic error resetting technique can reduce the complexity of our estimation with negligible accuracy impact, especially for circuits where long propagation paths exist.

3.4.5 Limited Depth Correlation Analysis

Due to the consideration of *pair-wise* signal correlations, the worst case complexity of CCM is quadratic $\binom{N}{2}$, i.e. $O(N^2)$, where N is the total number of gates. Although structural properties of the netlist can be explored to consider only correlated signals on reconvergent paths, its impact on scalability improvement is very limited. Therefore, to make our proposed

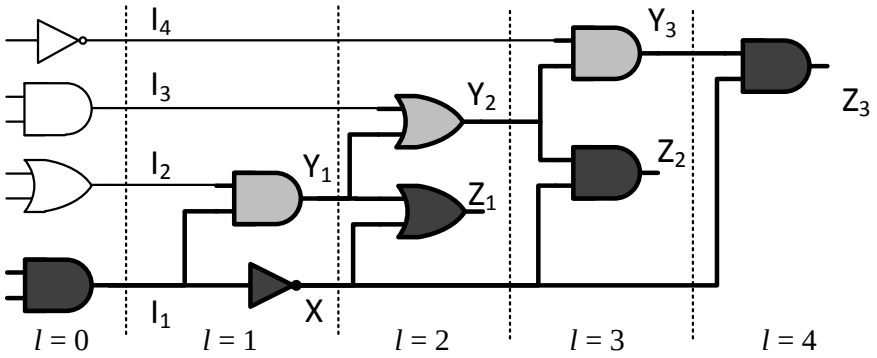


Figure 3.8: Correlation reduction with signal/error propagation

method applicable to large circuits (e.g. more than 5000 gates), a heuristic algorithm is necessary to improve the scalability while maintaining satisfactory accuracy.

Inspired by the switching activity estimation work [100, 103], one heuristic algorithm - *limited depth correlation analysis* is proposed and integrated into our method to further improve its scalability. The basic idea is to consider spatial signal/error correlations within only limited logic depth d. Intuitively, starting from the gate with multiple fanouts, the farther the gates on different propagation paths are from the fanout point, the less correlated the corresponding output signals will be. The rational is that the signals outside the reconvergent regions will interact with these fanout signals and statistically *decouple* the corresponding gate outputs. We take one simple circuit in Figure 3.8 to illustrate this phenomenon.

In this circuit there are three reconvergent regions: $\{I_1 \rightarrow Y_1 \rightarrow Z_1,\ I_1 \rightarrow X \rightarrow Z_1\}$; $\{I_1 \rightarrow Y_1 \rightarrow Y_2 \rightarrow Z_2,\ I_1 \rightarrow X \rightarrow Z_2\}$ and $\{I_1 \rightarrow Y_1 \rightarrow Y_2 \rightarrow Y_3 \rightarrow Z_3,\ I_1 \rightarrow X \rightarrow Z_3\}$. Assume all the signals at level 0 are uncorrelated, $p(I_1) = p(I_2) = p(I_3) = p(I_4) = 0.5$, and additionally, $p(I_{1_f}) = 0.5$ (i.e. the error probabilities of signal I_1 is 0.5). From the circuit structure, the logic functions of concerned signals can be derived as follows:

$$
\begin{aligned}
X &= \bar{I}_1, & X_f &= I_{1_f} \\
Y_1 &= I_2 I_1, & Y_{1_f} &= I_2 I_{1_f} \\
Y_2 &= Y_1 + I_3 = I_2 I_1 + I_3, & Y_{2_f} &= \bar{I}_3 Y_{1_f} = \bar{I}_3 I_2 I_{1_f} \\
Y_3 &= Y_2 I_4 = I_2 I_1 I_4 + I_3 I_4, & Y_{3_f} &= I_4 Y_{2_f} = I_4 \bar{I}_3 I_2 I_{1_f}
\end{aligned}
$$

Base on these Boolean functions and Equation (3.3), we can derive the exact joint signal/error probabilities and PCCs of concerned signals at the reconvergent points Z_1, Z_2 and Z_3:

$$
\begin{aligned}
\rho_{I_1,I_1} &= 1 & \rho_{I_{1_f},I_{1_f}} &= 1 \\
\rho_{X,Y_1} &= -0.5774 & \rho_{X_f,Y_{1_f}} &= 0.5774 \\
\rho_{X,Y_2} &= -0.2582 & \rho_{X_f,Y_{2_f}} &= 0.3780 \\
\rho_{X,Y_3} &= -0.1348 & \rho_{X_f,Y_{3_f}} &= 0.2582
\end{aligned}
$$

The calculated values show that with the signal/error propagation, the highly correlated signals (i.e. PCC is 1 at the fanout point) become less and less correlated (i.e. PCC approaches 0).

This trend holds not only for error-free signals, but also for erroneous signals. Therefore, using independent assumption for probability estimation of less correlated signals will cause smaller inaccuracy.

Based on this observation, the limited depth correlation heuristic is implemented and integrated into our method. As shown in Algorithm 3.1, the limited depth d is specified by the user, then for each gate g with multiple fanouts, a modified Depth First Search (DFS) procedure LIMITED_DEPTH_DFS (G, g, d) is carried out to identify the reconvergent regions within this depth. The gates in each reconvergent region will be recorded in one Boolean correlation matrix m and their outputs will be considered as correlated signals in the later error estimation phase. Note that in our implementation of the function Depth_First_Search (G, g, d) in line 16, *INV* gate is excluded from the depth calculation, as it has no logical masking effect for error propagation at all.

Algorithm 3.1 Limited Depth Correlation Analysis

```
1: Input: Netlist G, search depth d, Boolean correlation matrix m
2: Correlated_Regions = empty set
3: for each gate g in G do
4:     if fanout of g >1 then
5:         tmp_Correlated_Regions = LIMITED_DEPTH_DFS(G, g, d);
6:         Add tmp_Correlated_Regions to Correlated_Regions
7:     end if
8: end for
9: for each region r in Correlated_Regions do
10:    for each two gates g1, g2 in r do
11:        m[g1, g2] = true; ▷ These two gates should be considered as correlated in the error
       estimation
12:    end for
13: end for
14:
15: function LIMITED_DEPTH_DFS(G, g, d)
16:    revisiting_gates = Depth_First_Search(G, g, d)      ▷ DFS procedure within depth d
17:    if number of revisiting_gates >0 then
18:        backtrack and generate reconvergent paths;
19:        return reconvergent regions;
20:    end if
21: end function
```

This algorithm explores the structural properties of circuit netlist to identify the correlated gates with a single user-specified parameter: search depth d. Take Figure 3.8 as example, the starting point of DFS is gate I_1 (fanout node). When $d = 1$, the DFS visiting sequence can be $I_1 \rightarrow X \rightarrow Z_1 \rightarrow Z_2 \rightarrow Z_3 \rightarrow Y_1$. Note *INV* gate X is considered as transparent with depth increase of zero. No gate is revisited and no reconvergent region is found, therefore signal pairs Y_1 and X will be considered as uncorrelated in error estimation. When $d = 2$, the DFS visiting sequence can become $I_1 \rightarrow X \rightarrow Z_1 \rightarrow Z_2 \rightarrow Z_3 \rightarrow Y_1 \rightarrow Z_1 \rightarrow Y_2$ and Z_1 is revisited, therefore reconvergent region is identified. In this case, signal pairs Y_1 and X will

be considered as correlated when error probabilities should be calculated.

If not only the error probabilities but also the error correlations of primary output are of concern, we only need to add one *virtual* gate where all primary outputs converge, and Algorithm 3.1 can automatically take into account the necessary propagation paths to calculate primary output correlations.

Actually, the search depth d can either be specified by the user, or determined automatically by the algorithm. For example, this depth can be set to different value according to the number of fanout. The more fanout branches, the larger search depth will be used, therefore more reconvergent regions may be found. Again, such *variant* depth correlation search implies the flexibility of our proposed algorithm.

3.5 Extensions of the Proposed Method

As the proposed method addresses signal and error correlations explicitly in a unified way, it enables the extensions of this method to the following two directions.

3.5.1 Multiple Errors Propagation with Correlation

Our approach can be extended to account for more complex error models than single bit flip. As stated in [13, 104], multiple transient errors are no longer negligible in logic circuits. To obtain accurate estimation, correlated error occurrence and the correlations in the propagation of errors originating from different error sites must be taken into account, rather than the independent assumption used in existing technique [105].

Our proposed approach needs only minor modification to address these correlations. Specifically, as error occurrence of impaired gates is no longer deterministic and independent, both error probabilities of the error sites and their correlation coefficients can be specified by the user as plug-in parameters. In the case of multiple errors modeling, there are multiple error cones and the *unified error cluster* must be considered which is the union of the gates in the fanout cones of all error sites. The gates within the unified error cluster are treated as super gates and those at the boundary as semi-super gates, as illustrated in Figure 3.9. In this way, the same framework can be used to handle multiple error sites and their correlated propagation.

3.5.2 Block-level Error Estimation

Another advantage of the CCM method is that it explicitly calculates correlation coefficients from primary inputs to primary outputs. Therefore, it supports cascaded and hierarchical analysis of error propagation among different function blocks, which provides the possibility to raise the abstraction level from logic to register transfer level. However, two issues must be taken care of for this analysis.

First, it is essential to take into consideration the signal/error correlation of primary inputs. Actually, Algorithm 3.1 needs only small modifications to handle correlated inputs. For function LIMITED_DEPTH_DFS (G, g, d), the original single source DFS should be replaced with *multiple* source DFS, that is to say, the correlated primary inputs will be considered as

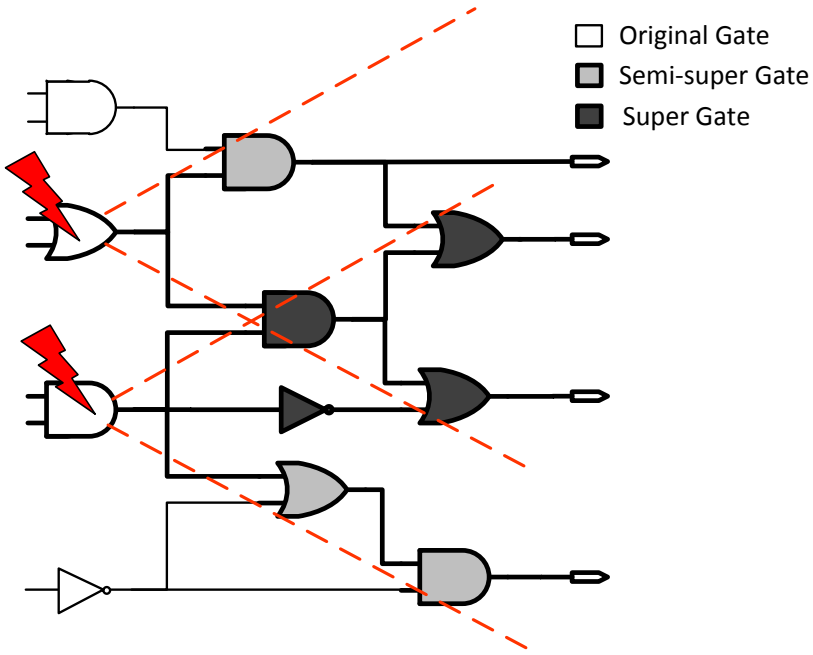


Figure 3.9: Propagation paths with multiple error cones

the start points of DFS. In this way, both kinds of spatial correlations mentioned in Section 3.3 are covered by the proposed approach, since CCM intrinsically supports conveying signal statistics from primary outputs of one block to the primary inputs of its successors. To simplify the correlation calculation without loss of much accuracy, similar techniques as dynamic blocking of error propagation in Section 3.4.4 can be employed as well to treat primary outputs with low error probabilities as error-free ones.

Second, the reconvergency among multiple blocks needs to be handled as well. The technique in [106] provides one promising solution. First the signal probability of fanout node F is set to 0 or 1, i.e. this node has deterministic value 0 or 1, thus the statistical dependency of fanout signals can be eliminated. After the two deterministic cases are handled separately, the output error probabilities can be obtained by combining the two cases together according to the signal probability of node F. The scalability of this technique is questionable for netlists with large number of reconvergent regions. However, at block level the number of modules for the entire circuit is relatively small, therefore this technique is suitable regarding the complexity.

3.6 Experimental Results

3.6.1 Experimental Setup

The proposed approach has been implemented in C++ using the igraph library [107], and experiments have been performed on a workstation with Intel E5540 2.53 GHz and 16 GB RAM.

The overall estimation and validation flow is illustrated in Figure 3.10. The circuit is first

synthesized with elementary gates, then the generated netlist is parsed and mapped to the corresponding graph structure. Both fault injection and the proposed Correlated Error Propagation (CEP) technique use this graph structure to evaluate the output error probabilities. After traversal of all error sites, the probabilities are compared to obtain the accuracy results.

In the experiments on benchmark circuits, the primary inputs of the circuits are assumed to be independent with signal probability 0.5. For the fault injection part, in the scenario of circuits with small number of primary inputs (<20), we use exhaustive simulation to obtain exact primary output error probabilities for all error sites, while for more primary inputs where exhaustive simulation is intractable, we turn to the statistical fault injection, based on Monte-Carlo simulation [108]. For the error probability of each primary output, with 95% confidence level, the Confidence Interval (CI) is given by

$$(\hat{p_e} - 1.96\frac{S}{\sqrt{N}}, \hat{p_e} + 1.96\frac{S}{\sqrt{N}})$$

where $\hat{p_e}$ is the estimated error probability of the primary output (i.e. mean estimation), S^2 is the variance estimation computed on the fly and N is the number of input vectors. Taking into consideration that the probability values are between 0.0 and 1.0, absolute width of the confidence interval

$$w_{CI} = 2 \times 1.96 \times \frac{S}{\sqrt{N}} \leq 0.005$$

is used as termination criteria.

In the proposed analytical method, graph preprocessing is carried out at setup phase and correlated gates within specified logic depth are identified. This step takes very short time because the search depth d is always very small (≤ 4). In this phase, error-free signal probabilities and correlation coefficients are also calculated to be used for later error estimation. Then for each error site, its error probability is set to 1.0 and its correlation coefficients with all the other signals at this level are set to 1.0 (independent error occurrence). After that the gates in the error cone are extracted and tagged as either super gates or semi-super gates. Starting from the logic level of error site, different formulas in Section 3.4.3 are used according to different gate types in the graph traversal. Finally, the error probabilities of primary outputs are obtained for later accuracy comparison.

Our proposed method has been validated for several ISCAS 85 benchmark circuits with regard to both accuracy and scalability. For the large combinational cores extracted from sequential circuit benchmarks ISCAS 89 and ITC 99, fault injection of each error site using simulation is not feasible. Therefore, only absolute runtime of our analytical method are reported to illustrate its scalability with different circuit sizes. In addition, to illustrate the necessity of error correlation modeling, we use the 32-bit ALU of OpenRISC 1200 processor [109] as a case study to investigate the correlated error probabilities at primary outputs. The realistic signal probabilities and correlation coefficients for primary inputs of this ALU are extracted from Value Change Dump (VCD) files of OpenRISC processor running Mibench benchmarks [110].

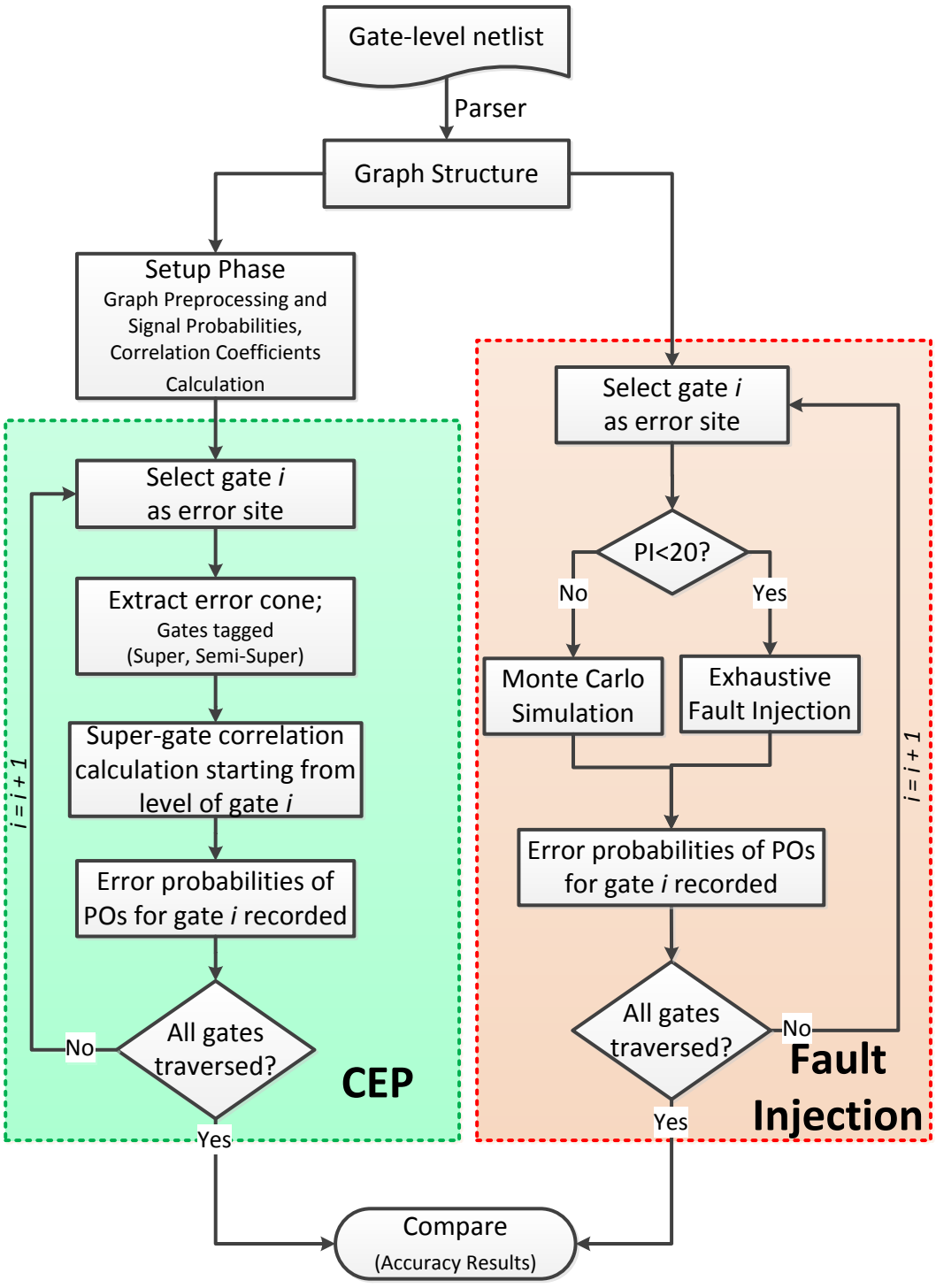


Figure 3.10: Error probability estimation and validation flow

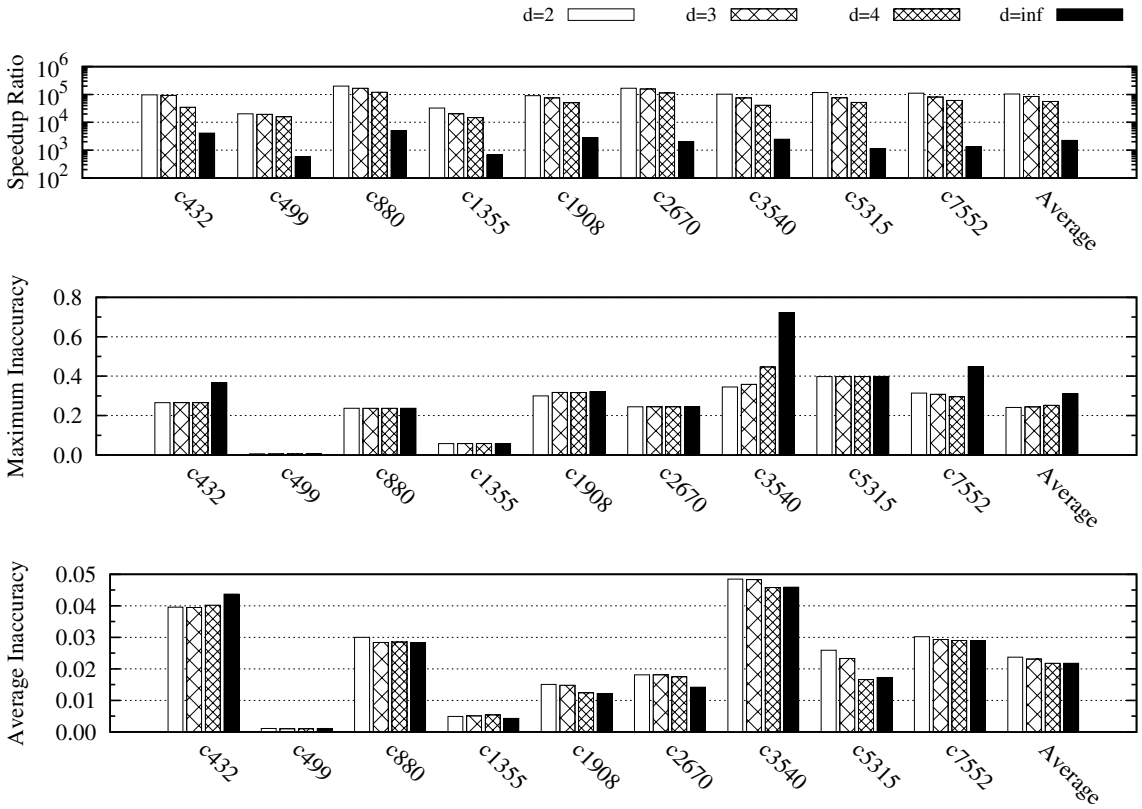


Figure 3.11: Speedup, maximum and average inaccuracy for considering variant correlation depths

Table 3.1: Runtime (sec.) and accuracy of proposed CEP approach with correlation depth $d = 2$

Benchmark	# of Gates		MC	4-value EPP [95] Inaccuracy				CEP Inaccuracy			
	Total Gates	Error Sites	Runtime	MAX	AVG	Runtime	Speedup vs CEP	MAX	AVG	Runtime	Speedup vs MC
c432	198	198	5412.5	0.301	0.031	0.004	13.2	0.265	0.039	0.06	90210
c499	575	575	11112.8	0.500	0.031	0.020	26.2	0.006	0.001	0.55	20205
c880	459	459	30885.1	0.825	0.099	0.007	22.9	0.237	0.030	0.16	193032
c1355	588	588	17587.5	0.273	0.020	0.023	23.1	0.058	0.004	0.54	32569
c1908	524	524	40892.9	0.657	0.040	0.016	26.8	0.299	0.015	0.45	90873
c2670	834	834	106617.0	0.825	0.049	0.015	40.9	0.244	0.018	0.64	166589
c3540	1088	100	15022.4	0.722	0.099	0.003	42.1	0.345	0.048	0.15	102807
c5315	1666	100	21425.2	0.872	0.052	0.002	58.5	0.398	0.025	0.16	133407
c7552	1999	100	28195.4	0.699	0.079	0.003	68.2	0.313	0.030	0.21	136606
s38584	11818	11818	-	-	-	74.040	1.4	-	-	105.15	-
s38417	12351	12351	-	-	-	81.002	1.4	-	-	113.03	-
b20	38912	38912	-	-	-	163.424	13.1	-	-	2149.73	-
b21	39168	39168	-	-	-	162.468	12.6	-	-	2053.11	-
average	-	-	-	0.630	0.056	-	27.0	**0.240**	**0.023**	-	**107366**

3.6.2 Benchmark Results

As our motivation is analyzing correlated error propagation from logic level to higher abstraction levels, individual error probabilities rather than an overall reliability metric are preferred. For each error site, the error probabilities of all primary outputs in the error cone are calculated first, and then they are compared one by one with fault injected Monte-Carlo simulation to obtain the accuracy results. For benchmarks with number of gate larger than 1000, the Monte-Carlo simulation takes very long time if *each* error site is simulated, so 100 gates are randomly selected for fault simulation. We report the maximum absolute inaccuracy (MAX) and average absolute inaccuracy (AVG), as defined in the following:

$$\begin{aligned} MAX_{inaccuracy} &= \max_{1\le n\le N} \max_{1\le i\le O_n} |Pe_{CEP}(PO_i) - Pe_{MC}(PO_i)| \\ AVG_{inaccuracy} &= \frac{1}{N \cdot O_n} \sum_{n=1}^{N} \sum_{i=1}^{O_n} |Pe_{CEP}(PO_i) - Pe_{MC}(PO_i)| \end{aligned}$$

where N is the number of error sites evaluated in the benchmark circuit, O_n is the number of primary outputs in the error cone of error site n and $Pe(PO_i)$ is the error probability of primary output i. The relative inaccuracy is deliberately excluded due to its misleading meaning for small probability values.

To illustrate the trade-off between accuracy and scalability discussed in Section 3.4.5, we also carried out the experiments with different correlation depth and the results are shown in Figure 3.11, in which *d=inf* means no depth limitation is considered. With regard to accuracy, for most of the benchmarks, the average inaccuracies are always reduced or almost the same when larger depth of correlation is considered. This means that with the logic depth increasing and correlation degree decreasing, the uncorrelation assumption causes less inaccuracy. At the same time, the speedup ratio of our analytical CEP approach with regard to Monte-Carlo simulation increases a lot, e.g. the average speedup ratio of *d=2* is approximately 2 orders of magnitude larger than that of *d=inf*.

However, for benchmark circuits *c432, c3540* and *c7552* the maximum inaccuracies for *d=inf* are even larger than that for *d=2*. After investigating the circuit structure, we find that it is due to the first-order approximation of CCM introduced in Section 3.3.1. After the propagation of errors on *multiple long* (e.g. fanout > 5) reconvergent paths, this first-order approximation results in larger inaccuracy compared to the case where the inputs of reconvergent gate are assumed to be independent. That is why the maximum inaccuracies of the benchmark *c432, c3540* and *c7552* with *d=inf* are larger than that with *d=2*. This interesting phenomenon illustrates that considering limited depth of correlation is beneficial not only to speedup our analytical method, but also to avoid inaccuracy accumulation of CCM for some special worst-case scenarios.

Table 3.1 shows the runtime and accuracy comparison between our proposed CEP ($d = 2$), Monte-Carlo simulation (MC) and the previously proposed 4-value Error Propagation Probability (EPP) technique [95]. It is worth mentioning that our reported inaccuracy values are based on *node by node* comparison of the error probabilities of all concerned primary outputs for all error sites, rather than one overall accuracy metric for the whole circuit. Taking this into consideration, the maximum inaccuracies are still in reasonable range, as shown similarly in the probability estimation work [100, 111]. Regarding runtime, compared with

Monte-Carlo simulation our proposed method with depth $d = 2$ exhibits different speedup ratios from 4 to 5 orders of magnitude, which reflect the complexity of CCM related to different circuit structures and additionally, the speedup potential of the proposed heuristics for various error cone structures.

Compared with 4-value EPP, our method has much smaller inaccuracy, less than half regarding both maximum and average inaccuracies. This is because the 4-value EPP considers only error correlations on reconvergent paths, while the signal correlations and cross correlations in Figure 3.2 are all ignored. This simplification makes it one order of magnitude faster than our method (for fair comparison, the time to obtain signal probabilities, 4-value EPP using time-consuming logic simulation while CEP using analytical CCM, are excluded in the runtime reported in Table 3.1), but causes larger inaccuracy for the estimated probabilities, especially for the highly correlated circuit structures. In addition to the significant accuracy improvement, the main advantage of our method is that we are able to provide correlated error information but 4-value EPP can not.

3.6.3 Case Study of OpenRISC 1200 ALU

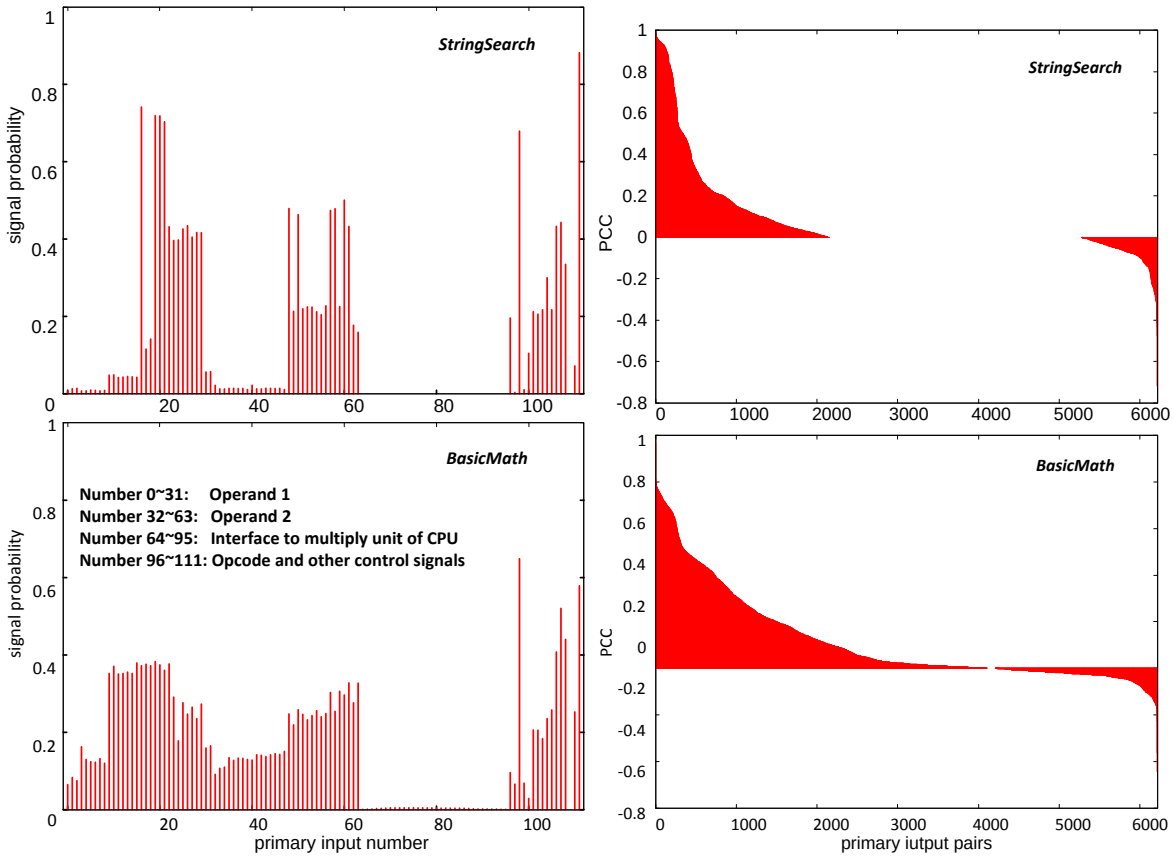
To illustrate the necessity of error correlation modeling, we investigated the ALU from OpenRISC processor as a case study. This ALU has 112 primary inputs including two 32-bit operands, one 32-bit multiply accumulation input, opcode, etc., and 38 primary outputs consisting of 32-bit computation results and some control signals. After synthesis there are 2854 elementary gates.

To investigate the error correlations of primary outputs under realistic workloads, we used the signal statistics extracted from VCD files as ALU primary inputs. The VCD files are dumped in the behavioral simulation of OpenRISC processor. Two typical applications - *StringSearch* and *BasicMath* from Mibench benchmarks [110] are selected to demonstrate the soft error statistics with workload dependencies. As shown in Figure 3.12, for these two applications both the signal probabilities and signal correlations have significant difference. As *StringSearch* executes mostly logic comparison operations without arithmetic multiplication, all of primary input numbers $64 \sim 95$ have signal probabilities 0.0 and the corresponding PCCs are also 0.0, i.e. signals with constant logic values are independent from all the other ones.

Using the proposed analytical CEP, we evaluated all possible error sites, which, however, is intractable with Monte-Carlo simulation. The error probability $Pe_{AVG}(i)$ of each primary output i averaged on all possible error sites is obtained first. We also calculated the pairwise joint error probabilities and Pearson product-moment Correlation Coefficient (PCC) to explicitly demonstrate their correlation degree. At the same time, the most interesting metric *conditional error probabilities* for primary outputs are also reported, which can answer the following question:
For soft errors occurring inside the functional block, given one primary output is erroneous, what is the probability that other outputs are erroneous at the same time?
The answer for this question is very useful for both high-level fault injection and efficient exploration of reliability enhancement.

The conditional error probability is defined as following:

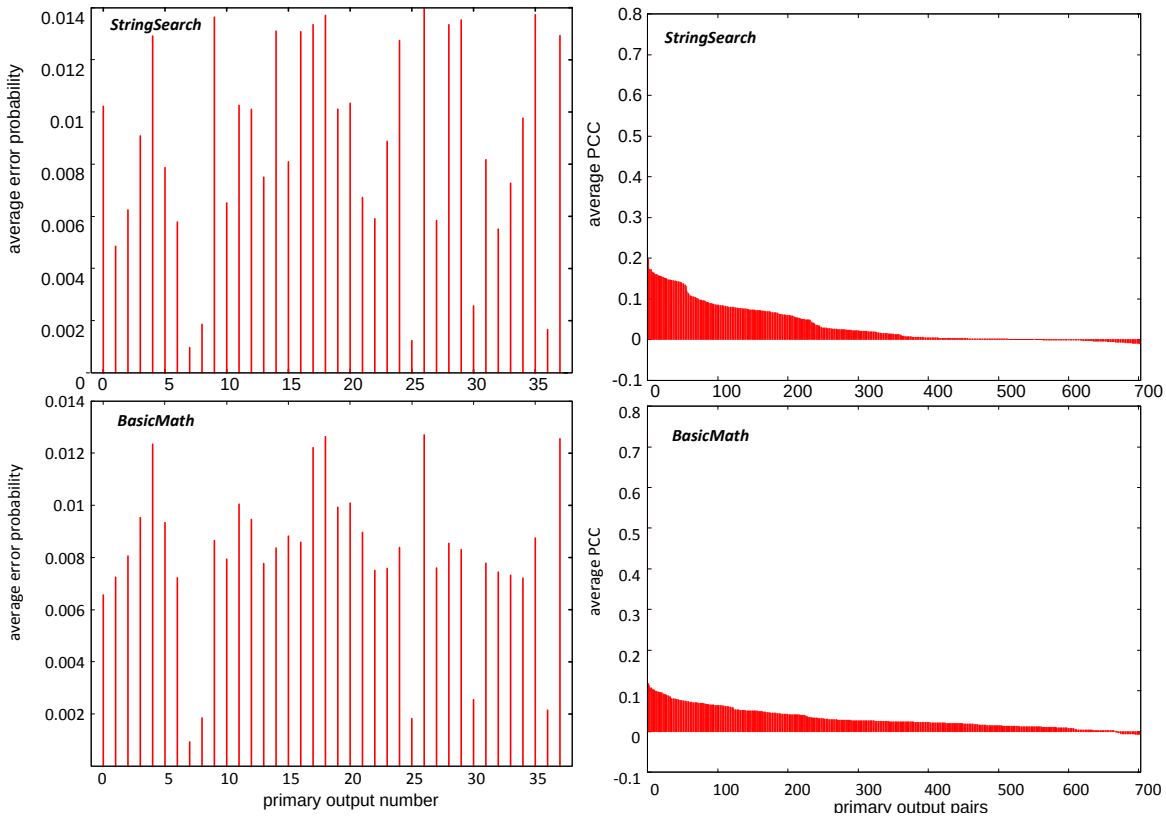
$$Pe_{AVG}(i|j) = \frac{Pe_{AVG}(ij)}{Pe_{AVG}(j)} \tag{3.14}$$



(a) Signal probabilities

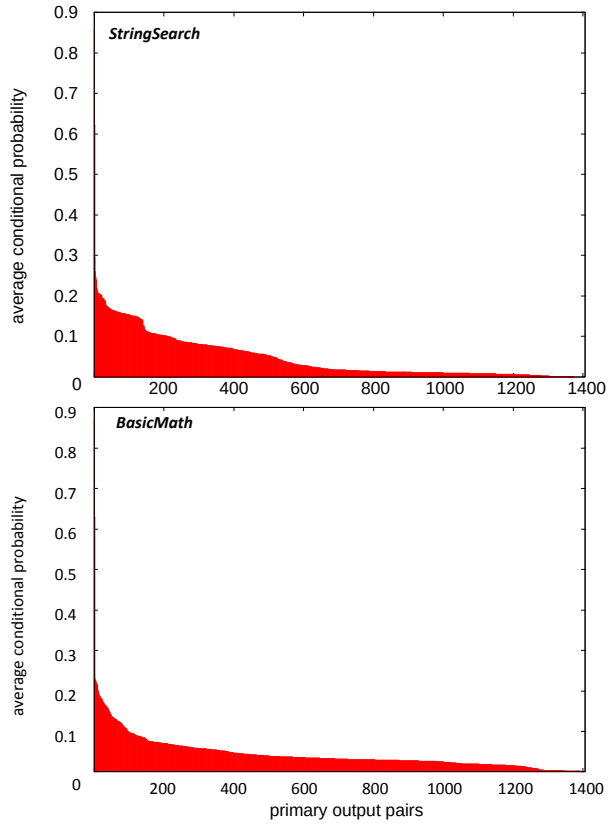
(b) Sorted pairwise PCCs

Figure 3.12: Primary input statistics of OpenRISC ALU running applications *StringSearch* and *BasicMath*



(a) Error probabilities

(b) Sorted pairwise PCCs



(c) Sorted conditional error probabilities

Figure 3.13: Error statistics for primary outputs of OpenRISC ALU running applications *StringSearch* and *BasicMath*

Note that $Pe_{AVG}(i|j)$ is different from $Pe_{AVG}(j|i)$, so there are totally $38 \cdot 37 = 1406$ output pairs. Although it is possible to approximate second order correlations using Equation (3.6), there will be $\binom{38}{2} \cdot (38-2) = 25308$ coefficients. For brevity it is sufficient to use pairwise correlations and conditional probabilities to illustrate the concept of *correlated bit flips*.

Figure 3.13(a) shows the average error probabilities of primary outputs of ALU running *StringSearch* and *BasicMath* applications. The average error probabilities of 32-bits computation results do not have large fluctuations, while the five outliers with particularly small error probabilities are listed in Table 3.2. From this table we can see that they are three control signals *cy_we, ov_we, flag_we* and two data signals *cyforw, ovforw*. Actually these three control signals become erroneous only when ALU executes wrong operations, and the average on all error sites including these executing operations on operands, makes the error probabilities of control outputs very small. For these two data signals, as the carry and overflow bits are generated in specific *ADDITION* operation and operands, the errors on these two signals are more probable to be masked, e.g. two erroneous operands are very likely to generate the same overflow bit '0' as two error-free operands, because *ADDITION* overflow occurs relatively rare. Due to this reason, the average error probabilities of these two data signals are also very small.

Figure 3.13(b) and 3.13(c) show that the PCCs and conditional error probabilities exhibit significant variations among all primary output pairs. The detailed experimental reports in Table 3.3 show that for both applications, the first two largest conditional error probabilities correspond to the same pair of outputs: carry write-enable signal and overflow write-enable signal, i.e. the carry write-enable signal has high probability to be erroneous given overflow write-enable signal is flipped (when the ALU executes the wrong operation due to internal soft errors), and vice visa. This observation is consistent with the functionality analysis of ALU, because normally these two signals are both '1' when the operation is *ADDITION*, otherwise they are both '0'. Our method can provide quantitative correlation values rather than the pure qualitative functional analysis.

For the remaining eight large conditional error probabilities in Table 3.3, almost all of them are related to the overflow write enable signal. However, for the two applications most of the signals having high error probabilities given *ov_we* is erroneous are different. This significant difference is due to the workload dependencies as shown in Figure 3.12. Different primary inputs statistics would generate different internal signal probabilities and correlations, therefore result in different masking effects in the error propagation.

In a word, our quantification of error correlations can not only identify highly correlated error signals due to the intrinsic functionalities, but also capture the influence of different workloads on error propagation and correlations. These quantified correlations and conditional probabilities can provide valuable insights and guides for the fault injection based simulation or analytical error estimation at higher abstraction levels.

3.7 Conclusion and Summary

Explicitly addressing the correlated bit-flips propagating from low level circuits is essential for accurate error estimation and error abstraction at high levels. This chapter described a novel approach to explicitly take into account both signal and error correlations in a unified way, therefore it can provide quantified error correlations. Based on the concept of error propa-

Table 3.2: Outliers of average error probabilities for primary outputs of OpenRISC 1200 ALU

StringSearch		BasicMath	
signal i	$Pe_{AVG}(i)$	signal i	$Pe_{AVG}(i)$
cy_we	0.0026	cy_we	0.0026
ov_we	0.0019	cyforw	0.0022
cyforw	0.0017	ov_we	0.0019
ovforw	0.0012	ovforw	0.0018
flag_we	0.0009	flag_we	0.0009

cy_we: carry write enable; ov_we: overflow write enable; cyforw: carry forward
ovforw: overflow forward; flag_we: comparison flag write enable

Table 3.3: Top 10 conditional error probabilities for primary outputs of OpenRISC 1200 ALU

StringSearch		BasicMath	
signal pair $i\|j$	$Pe_{AVG}(i\|j)$	signal pair $i\|j$	$Pe_{AVG}(i\|j)$
cy_we|ov_we	0.8572	cy_we|ov_we	0.8689
ov_we|cy_we	0.6229	ov_we|cy_we	0.6302
result[16]|ov_we	0.2958	result[0]|ov_we	0.2331
result[13]|ov_we	0.2598	result[3]|ov_we	0.2272
result[8]|ov_we	0.2470	result[2]|ov_we	0.2250
result[4]|ov_we	0.2407	result[7]|ov_we	0.2230
result[7]|ov_we	0.2392	result[1]|ov_we	0.2189
result[9]|ov_we	0.2288	result[8]|ov_we	0.2174
result[14]|ov_we	0.2183	result[5]|ov_we	0.2163
result[16]|cy_we	0.2150	result[4]|ov_we	0.2159

gation function, a new super-gate representation was conceived to address error probability and correlation problems with signal probability and correlation techniques. Experimental results showed our approach is 5 orders of magnitude faster than Monte-Carlo simulation, while the average inaccuracy of error probability estimation is only 0.02. This method can be extended to model correlated error propagation due to multiple error sites at logic level as well as hierarchical and modular error analysis at higher levels.

4 Vulnerability Analysis at Register Transfer and Behavioral Levels

As last chapter covers the logic level error analysis with detailed consideration of error correlation, in this chapter we abstract these error information to RTL and higher behavioral level, and introduce our error analysis framework by leveraging the word-level semantics. At RTL, according to the different error propagation properties, the data paths are analyzed with analytical formulas of error propagation probability, while the control paths are modeled as a state transition system, and formal probabilistic model checking is adopted to quantify the soft error vulnerabilities of the registers in the control modules. In addition, we also introduce how to evaluate the vulnerabilities of variables and operations at behavioral level with similar model checking techniques.

4.1 Introduction

The circuit level soft error analysis techniques [20, 50, 112–115] can provide very detailed and accurate estimation on the SER of each gate, but they can only be applied to the flattened netlists, and the useful high level behavioral semantics are ignored. For instance, at circuit level the registers with large bitwidth are treated as individual flip-flops and protected at *bit-level* using techniques such as DICE [61] and BISER [38]. However, at RTL or higher abstraction levels they can be protected together at *word-level* by ECC with less area and power overhead. Thus, performing the soft error evaluation at higher levels than circuit level has the potential to achieve more cost-efficient error mitigation.

An RTL design typically consists of a data path and a control path. In the data path, most operations are arithmetic and logic ones without control loops, hence the error propagation analysis can be performed with analytical approaches in one pass from primary input to output. However, in the control path, the error propagation and masking analysis is much more challenging due to the complicated control structures. In addition, compared with data operands, the control signals are very probable to manifest non-uniform probabilities due to the different running workloads [116]. This means for an accurate soft error analysis in a control path, the workload dependency has to be carefully taken into consideration.

Accurate evaluation of register vulnerabilities is particularly important for the control path, because it determines when and which RTL operations in the data path should be executed, hence is very critical to the design functionality. Moreover, recent research [34, 117] showed that in multi-processor SoCs, the so called MPSoCs, the complexity of control path and uncore logic increases significantly, which mandates efficient methods to investigate their vulnerabilities to provide cost-effective protection schemes. Although the occupied area of control path is not as high as data path, full register protection is still very costly and unnecessary [6, 53].

In addition, the timing impact of full register hardening can be another critical issue. For the well-known hardening techniques like DICE [61], Single Event Upset Tolerant [62] and Reinforcing Charge Collection [63], the delay of hardened registers can be 5%∼10% higher than the unprotected ones. As the control path determines the sequence of data transfer in data path and is critical to the system functionality, the delay impact due to full register hardening can be also significant. Therefore, the selective hardening scheme based on an accurate evaluation of register vulnerabilities is indispensable.

There are a few techniques focusing on soft error evaluation at RTL, including statistical fault injection [67] and analytical methods [5, 6, 69, 118]. The fault injection technique [67] is based on simulation and only covers a small portion of the state space. Therefore, it is incomplete and requires a long simulation time to obtain results with a reasonable accuracy [119]. Formal analytical techniques [5, 6, 69] are introduced in the soft error investigation due to the advantage in the completeness of the analysis. However, the techniques in [6, 69] only consider *qualitative* error metrics (i.e. not a quantitative probabilistic metric), while the work in [5] cannot model the workload dependency during error evaluation. In addition to these formal techniques, another RTL analytical method in [118] uses empirical hierarchical models to estimate the SER of the combinational data paths. However, it is inapplicable to the sequential control paths.

In addition to reliability characteristics of the RTL components, the behavioral variable and operation vulnerability, i.e. the probability of system failure given that component being erroneous, is another essential factor which influences the efficiency of selective protection and hardening. To evaluate these vulnerabilities, several important factors should be taken into account, including lifetime, functional dependencies, weight in conditional branches and error masking effects [120–124]. The work in [120–122] did not consider the important error masking phenomena, especially for the relational and logic operations which are common in the control flows. The authors in [123, 124] used path enumeration to exploit program-level error propagation and masking mechanisms. However, path enumeration is not applicable for the control-intensive designs, as not only the number of paths grow exponentially with the number of branch nodes, but also the presence of loops makes this enumeration intractable.

In this chapter, we propose novel methods based on formal probabilistic model checking and analytical error propagation probability formula to *quantitatively* evaluate the component vulnerabilities at both RTL and behavioral levels. In particular, the main contributions are threefold:

1. We model the probabilistic behaviors of the RTL control paths, the random soft error occurrence and their interactions using formal Markov chains. In this modeling process, the workload dependency can be also taken into account. In addition, leveraging the RTL behavioral semantics, several state abstraction and model simplification techniques are proposed. They can significantly improve the scalability of the formal model checking in the register vulnerability evaluation;
2. By analyzing and deriving EPP rules for different RTL operations in the data paths, our analytical method provides a good trade off between accuracy and complexity.
3. With the design functionality represented as CDFG at behavioral level, we perform a comprehensive variable and operation vulnerability evaluation with similar model checking technique as in RTL.

The experimental results show that the proposed methods are capable of handling complex control modules in a typical embedded processor, and the EPP rules for representative data paths are with rather small inaccuracy. In addition, we successfully quantify the non-uniform vulnerability distributions among the RTL registers under different running workloads.

The organization of the rest of this chapter is as follows. Section 4.2 introduces the preliminary and problem statement of this work. Section 4.3 describes the proposed model checking method to evaluate the register vulnerabilities in RTL control paths. Section 4.4 introduces the word-level soft error estimation methodology in RTL data paths. Section 4.5 presents the modeling and evaluation technique to evaluate the vulnerabilities of behavioral variables and operations. Section 4.6 shows the experimental results, and finally Section 4.7 concludes this chapter.

4.2 Preliminary and Problem Statements

4.2.1 RTL Control and Data Paths

An RTL design typically consists of two basic components: a control path and a data path [39, 125]. As shown in Figure 4.1, the data path receives the data, performs the data manipulation and generates the output results. In contrast, the control path identifies the status of the RTL design and provides control signals to the data path, determining when and which RTL operations should be executed. The control path is typically modeled as a Finite State Machine (FSM), containing the state registers, the next-state function and the output function.

For the error evaluation, in the data path the error propagation analysis can be performed with analytical approaches, as the typical register transfer operations have regular error masking properties. Furthermore, the operands in the data path can be generally assumed to have uniform signal probabilities [126].

However, in the control path the error propagation properties are significantly different from that in the data path. As shown in Figure 4.1, with prevalent logical operations and branch constructs, the control path has more sophisticated structures than the data path with regular operations. The error propagation in such control path can not be assumed transparent any more. To make things more complicated, the RTL control signals are very probable to manifest non-uniform probabilities [116]. That is to say, accurate error analysis in the control path requires carefully considering both the sophisticated error masking effects and various probabilistic distributions of the control signals.

4.2.2 Fault Model and Register Vulnerability

The fault model adopted in this RTL evaluation work is Single Event Upset (SEU), i.e. a bit-flip in the impaired register. Nevertheless, our evaluation method can be also extended to handle Multiple Bit Upset (MBU) by specifying simultaneous multiple bit flips during the modeling of error occurrence, as described later in Section 4.3.1.

In this work, the RTL *registers* refer to the data storage elements which typically correspond to circuit level flip-flops in the synthesized netlist. In our soft error evaluation, the RTL registers are treated either at bit level (single-bit control register, e.g. data valid bit) or at

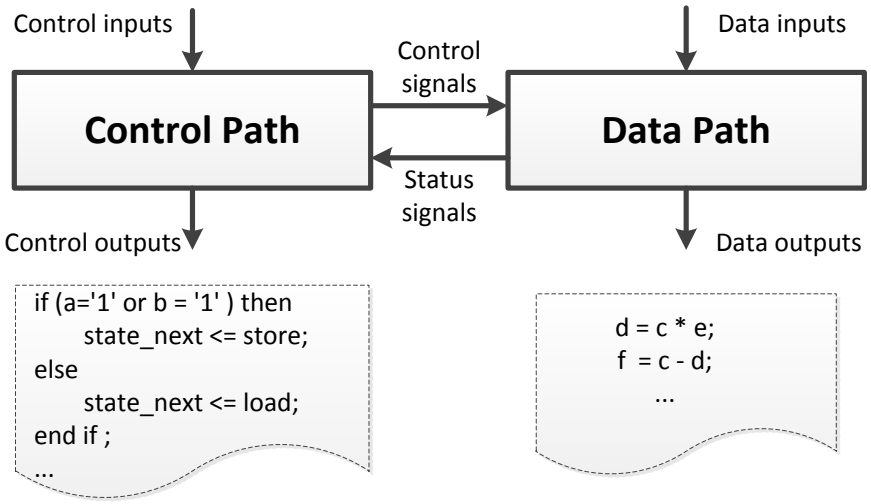


Figure 4.1: RTL control and data paths with code examples

word level (multiple-bit register, e.g. 32-bit address). Additionally, we use the term RTL *signals* to represent both the internal registers (REGs), the primary inputs (PIs) and primary outputs (POs).

The *vulnerability* of a register is defined as the probability of having a system failure, given an SEU occurs in this register. It is a conditional probability which reflects the error sensitivity of the RTL design to this specific register. Without loss of generality, in this work *system failure* is defined as error being observed in at least one of the RTL POs. With the flexibility of model checking, the system failure definition can also be adjusted according to the user specification.

4.2.3 Formal Methods for Soft Error Analysis

To evaluate the SEU effects at RTL, both simulation-based fault injection [67] and formal methods [5, 6, 69] can be employed. A brief comparison of these two kinds of techniques is illustrated in Table 4.1. The simulation-based fault injection has fair scalability, but it suffers from incomplete input space coverage and relatively low accuracy. In contrast, the formal methods feature exhaustive search of the design space and generation of the evaluation results with high confidence. Its main disadvantage is that scalability improvement techniques have to be developed to handle large designs.

Table 4.1: Comparison between simulation-based fault injection and formal methods [6]

Aspect	Simulation-based fault injection	Formal methods
Input coverage	Not exhaustive	Exhaustive
Fault coverage	Random cycle to inject fault	Exhaustive
Confidence	Long simulation time to reach acceptable confidence	Very high
Scalability	Fair (accuracy may degrade)	Large designs need improvement techniques

The RTL control paths can be naturally modeled as state transient systems. They do not have as huge state space as the data paths with large bitwidth operands (e.g. 32 or 64 bits). Therefore the exhaustive feature of the formal methods can be fully explored to handle the complicated error analysis in the control paths [6].

Regarding previous work using formal methods at RTL, the authors in [69] use a formal verification approach to analyze the effectiveness of online error detection and correction logic. A model-checking based technique is employed in [6] to identify the registers that must be protected to guarantee correct system functionality, but it only provides a binary classification (yes/no) rather than a quantitative metric. Another formal method based on Boolean Satisfiability (SAT) formulation is introduced in [5] to analyze the soft error rate at RTL. It computes the error propagation probability as the ratio between the number of satisfying SAT instances and the number of all input combinations, however, it fails to capture the dependency of error probabilities on workloads.

In this work the Probabilistic Model Checking (PMC) [127] is employed to obtain the soft error vulnerabilities of RTL registers. PMC is a formal technique for modelling and analyzing the state transition systems that exhibit probabilistic behaviours. Compared with the formal technique in [6] which targets absolute guarantee of the functional correctness (i.e. *qualitative* metric), PMC takes the stochastic behaviors of the design into consideration, therefore it can provide probabilistic evaluation (i.e. *quantitative* metric). In addition, different from the SAT-based technique in [5], PMC accepts user-specified input statistics (e.g. non-uniform signal probabilities and correlations) in the modeling process, so the workload dependency can be accurately captured in the evaluation.

In a typical PMC process, the systems is modelled as Markov Chains, i.e. the stochastic process with the property that the next system state depends only on the current state rather than the sequence of states preceding it. While the Continuous-Time Markov Chains (CTMCs) use a dense time model and the state transitions can occur at any (real-valued) time instant, the transitions in Discrete-Time Markov Chain (DTMC) proceeds in discrete time steps. Therefore, the DTMCs can provide appropriate system model with discrete time units, such as the clock ticks in the modeling of RTL circuits [128].

In addition to the probabilistic system model, PMC also accepts a wide range of user specifications, which are formally described by probabilistic temporal logics [127]. These logics specify how the behaviour of a system changes over time and one of its important use is the representation of system properties to be checked by a model checker. According to the different types of temporal properties, the model checker can return either Boolean or numerical value, e.g. the property $P_{>P_e}$ [$\Diamond$ $fail$] means *"whether the system eventually fails with the probability larger than the predefined value P_e?"* and $P_{=?}$ [$\Diamond$ $fail$] is interpretated as *"what is the probability that the system eventually fails?"*. Here $\Diamond$ is a temporal operator specifying the "eventually in the future" behavior of the state transition system, and *fail* is a state property which can be represented by logic proposition, e.g. system states with non-zero value of the error flag signal, $value(e_f)! = 0$.

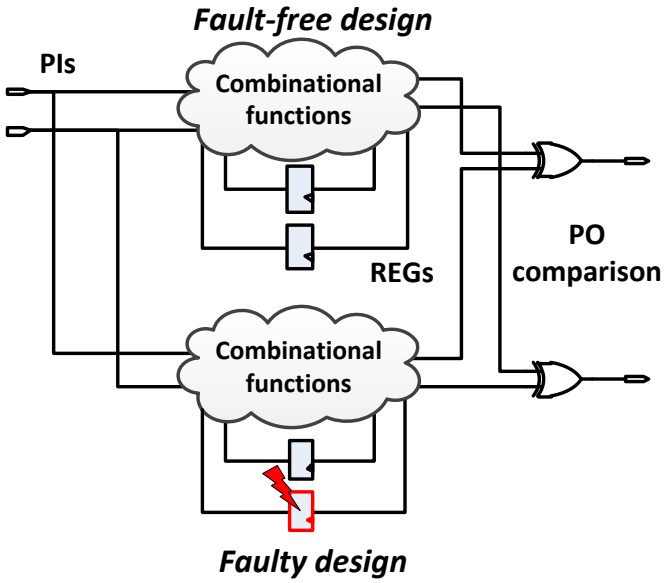


Figure 4.2: Structural view of the RTL error evaluation

4.3 Register Vulnerability Evaluation in RTL Control Paths

The PMC process for our register vulnerability evaluation can be generally divided into three phases:

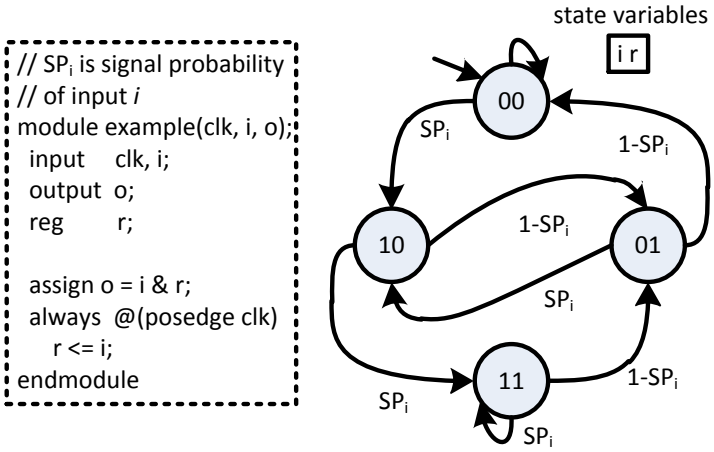
- *Model Construction*: The state transition model, which combines the probabilistic behaviors of an RTL design and the random soft error occurrence, is constructed;
- *Property Specification*: The temporal properties which can guide the model checker to extract the error probabilities at POs are specified;
- *Model Checking*: Model checker uses numerical methods to explore the state space of the constructed model, and finally gives the desired error probabilities.

4.3.1 Probabilistic Model Construction

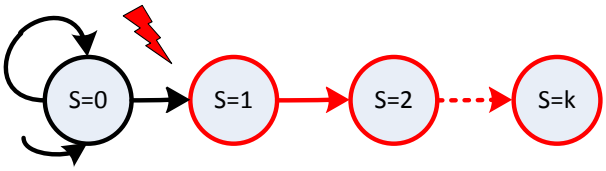
As mentioned earlier, an RTL control path can be modeled as an FSM and each time step corresponds to a state transition. Here one RTL state is represented by the unique values of the REGs and PIs, therefore for an RTL design with n PIs and m REGs[1], in worst case the state space size is 2^{m+n}. Based on the values of PIs and REGs in current time step, the next-state and output functions determine the values of REGs in the next time step and also the values of POs. These functions typically correspond to the combinatorial logics and are represented by Boolean expressions at behavioral RTL. Figure 4.2 shows the basic structure of an RTL design, and in addition, a faulty design to compare with the fault-free one for soft error evaluation.

To consider the probabilistic behaviors of an RTL design, we employ the DTMC model in which each state transition in the FSM is associated with a probability value. One DTMC state represents the value assignments to a set of *state variables* (corresponding to the PIs and the

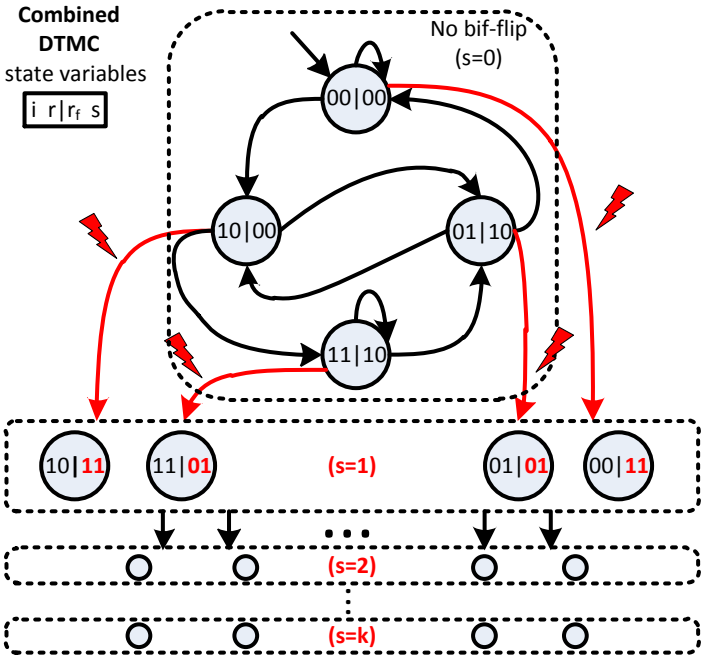
[1]For the estimation of state space size, the number n and m should be the bit-width sum of the RTL signals.



(a) A simple example of an RTL design and the corresponding fault-free DTMC



(b) Soft error occurrence DTMC



(c) Combined DTMC with soft error in register r

Figure 4.3: Model construction for the soft error evaluation

REGs in the RTL design). Figure 4.3(a) is a simple DTMC example, where the state variables are i and r, and the value of the output o is just a Boolean function of these two state variables. To capture the workload dependency in the constructed model, the signal probabilities of all PIs as well as the correlations can be integrated into the state transition probabilities. Note that only the PI probability distributions, the output and next-state Boolean functions need to be provided, then the REG and PO probabilities will be automatically computed by the model checker.

Modeling of SEU

After the fault-free RTL design modeling, the random occurrence of a soft error at any possible DTMC state needs to be integrated. We use the parallel composition mechanism of DTMC to model this effect. For one DTMC module composed with several parallel submodules, the global states of the entire DTMC are the interleaved states of each submodule. In this way, the occurrence of a bit-flip (Figure 4.3(b)) is interleaved with each possible state in the fault-free design (Figure 4.3(a)). As shown in Figure 4.3(c), in the combined DTMC there are two additional variables r_f and s. The variable r_f represents the possible erroneous value of the register r in the faulty design, and s is the state indicator of the bit-flip occurrence. In the states with $s = 0$ no bit-flip has occurred yet, and the states with $s = i$, $(i = 1, 2...k)$ correspond to the successive time steps after the bit-flip occurrence.

Modeling of MBU

To extend the SEU modeling to MBUs, the main modification is that during the transitions from states with $s = 0$ (i.e. error-free) to states with $s = 1$ (i.e. error occurrence), instead of only a single bit difference in the case of SEU, for modeling MBU the value of multiple bits are flipped. This is shown by the example in Figure 4.4, where the state variables r^1, r^2 and r_f^1, r_f^2 represent the error-free and possible erroneous values of the two registers, respectively. After modeling the MBU occurrence, the error propagation behavior is analyzed in a similar manner as SEUs. As the same set of state variables are already considered during the SEU analysis, the overall state size of modeling MBUs is the same as that of SEUs.

4.3.2 Property Specification

To obtain quantitative measurements of the modeled system, we need to specify the model checking properties expressed with probabilistic temporal logic.

For evaluating the soft error vulnerability of the impaired RTL register, a traversal of all paths in Figure 4.3(c) starting from the initial state and reaching the states with system failure is necessary. As defined in Section 4.2.2, it means in these failure states, there exist different PO values in fault-free and faulty RTL designs. As explained in Section 4.2.3, this kind of path traversal can be specified by the following temporal property

$$P_{=?}\,[\,\Diamond\, fail] \quad \triangleq \quad P_{=?}\,[\Diamond\,(s == k\,\&\,PO_e! = PO)] \tag{4.1}$$

where $P_{=?}$ means that the model checker should obtain the quantitative probability value, and the temporal operator $\Diamond$ means starting from the initial state, eventually the states with

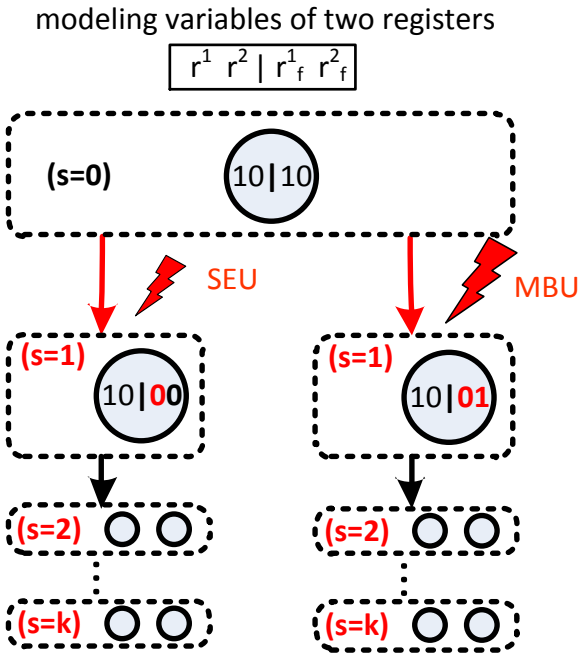


Figure 4.4: Example of modeling the SEU and MBU

system failure will be reached. The *fail* state property is defined as $(s == k \,\&\, PO_e != PO)$, which means that at k steps after SEU occurrence, the primary outputs from fault-free and faulty DTMC are different.

Therefore, this property will guide the model checker to obtain the overall probability that, after an SEU occurs, the RTL system proceeds k time steps and eventually encounters a system failure. The failure probability obtained here is a conditional probability given a bit-flip occurs in the specific register, i.e. register vulnerability. Note that taking advantage of the flexibility of property specification, other error scenarios can also be analyzed by the model checker:

- The system failure definition may be adjusted according to the RTL semantics. For instance, when one “validation bit" signal is inactive, the errors in data outputs do not matter and can be safely ignored;
- there exist correct PO values but erroneous REG values in the faulty RTL design, i.e. latent errors;
- the PO/REG errors appear at the k^{th} step $(s = k)$, or within k steps $(s \leq k)$ after the SEU occurrence.

4.3.3 Model Checking

We use PRISM [129], a symbolic model checker to rigorously analyze the DTMC model. The tool has been widely used for quantitative verification in a broad spectrum of application domains. It supports the construction of various types of probabilistic models and adopts the compact binary decision diagrams to efficiently manipulate them. In this work PRISM is also combined with Sigref [130], a tool that performs the bisimulation-based state minimization [127] for faster model checking.

State transition systems generally contain large degree of redundancy as there are many states which behave very similarly. A bisimulation $\mathcal{R}$ on the state space is a binary relation that relates states which are indistinguishable in some sense. The idea of bisimulation minimization is to collapse all states that are related by a bisimulation $\mathcal{R}$ into one meta-state, which has the same behavior as all contained states. It can be shown that bisimilar states satisfy the same formula of the probabilistic temporal logic [131]. Therefore, bisimulation minimization preserves all properties expressed with this temporal logic while reducing the size of the model. Consider the two states s_1 and s_2 as an example in Figure 4.5. Although they have different outgoing probability distributions, the cumulative probability of going to states satisfying the same property, e.g. *system fails*, which are indicated by the grey color is 0.7 for both states , so the two states are indistinguishable with regard to the concerned property and thus can be merged.

Bisimulation minimization has been proved to speed-up PMC runtime for many systems [132]. For large systems it can even enable the analysis that could not be dealt with before [133]. Furthermore, the technique has the advantage of being fully automated and requiring no manual interaction with the user.

4.3.4 Scalability Improvement of the RTL Model Checking

Formal model checking techniques exhaustively explore the entire state space to obtain the desired results with high confidence. However, the well-known state explosion problem limits their applicability and hence scalability improvement techniques are essential. In our evaluation of soft error probabilities, the size of state space has an exponential relation with the numbers of PIs and REGs in the RTL design. Therefore, our main goal is to reduce these two numbers in the model construction phase.

Except for adopting the state abstraction technique - data type reduction [134, 135], we propose another two new techniques to further improve the scalability, i.e. verification-assisted model simplification, and time-multiplexed PIs partitioning. They can either remove the modeling redundancy by RTL abstraction and simplification, or explore the structural decomposition of the state space. As the error propagation properties are preserved in these processes, applying them do not influence the evaluated vulnerability values.

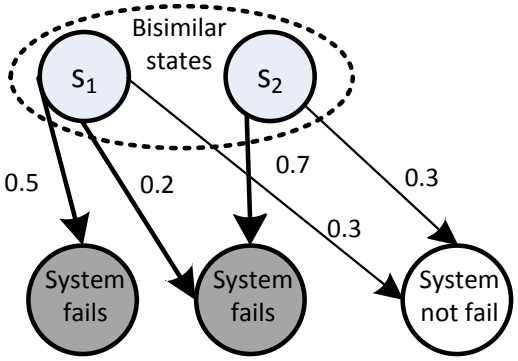


Figure 4.5: Example of bisimulation equivalent states

Data Type Reduction (DTR)

One main advantage of the error investigation at RTL is that the word-level information can be leveraged to reduce both the numbers of PIs and REGs. For a given property to analyze in the model checking, RTL signal with large bitwidth can be treated as several multi-bit blocks [134], as opposed to clusters of single bit signals at circuit level without any special semantics. This kind of abstraction is particularly suitable for the RTL control paths, in which there are limited number of signals with large bitwidth (e.g. *data, address*). These signals always move around as blocks in the RTL designs.

For some RTL arithmetic and block-assignment operands, the bit-width of concerned operands can be significantly reduced, while the error propagation properties are still preserved. According to the bits extraction and concatenation operations in RTL designs, the entire multi-bit signals may be divided into several segments. Take the following codes as example:

```
saved_addr_r[3:2] <= saved_addr_r[3:2] + 1;
...
saved_addr_r[31:12] <= start_addr[31:12];
```

The errors on the right hand side of these two assignments propagate directly to the left hand side. According to the involved operations of the different segments in the register *saved_addr_r*, its total 32-bits can be broken into two segments: [31:12] and [11:0]. The first segment is related to the assignment of the signal *start_addr*, and for the second segment, due to the transparent error propagation any errors in this segment will stay there even after the assignment is finished. Both segments can be abstracted to a single bit, when similar treatments are performed for the signal *start_addr*. This abstraction modifies the original functionality, but preserves the error propagation properties. This powerful abstraction of error propagation at word-level is only possible when the analysis is done at high levels such as RTL. In addition, this abstraction is independent of the word width, e.g. even when the width of *saved_addr_r* signal increases to 64 or 128 bits, the similar abstractions are still applicable. Our experimental results in Section 4.6 show that, on average it can reduce the number of PI and REG variables in the model construction by more than 70%.

Verification-assisted Simplification (VAS)

Another efficient way of scalability improvement is to explore the local influence of error propagation, and remove the unnecessary REGs in the model construction process. As illustrated in Figure 4.6, to check the correctness of PO values the fault-free and faulty versions of the RTL design need to be compared, which implicitly doubles the number of internal REGs. However, after the bit-flip occurs and in the following error propagation, the registers influenced by the errors are always in limited regions of the entire design. Due to the similar functionalities of fault-free and faulty designs, straightforward comparison will introduce significant modeling redundancy.

To obtain a compact error propagation model, we propose to employ Sequential Equivalence Checking (SEC) [136] to merge equivalent registers for the removal of modeling redundancy. The key point here is how to incorporate the *transient* behavior of the soft error in the faulty design.

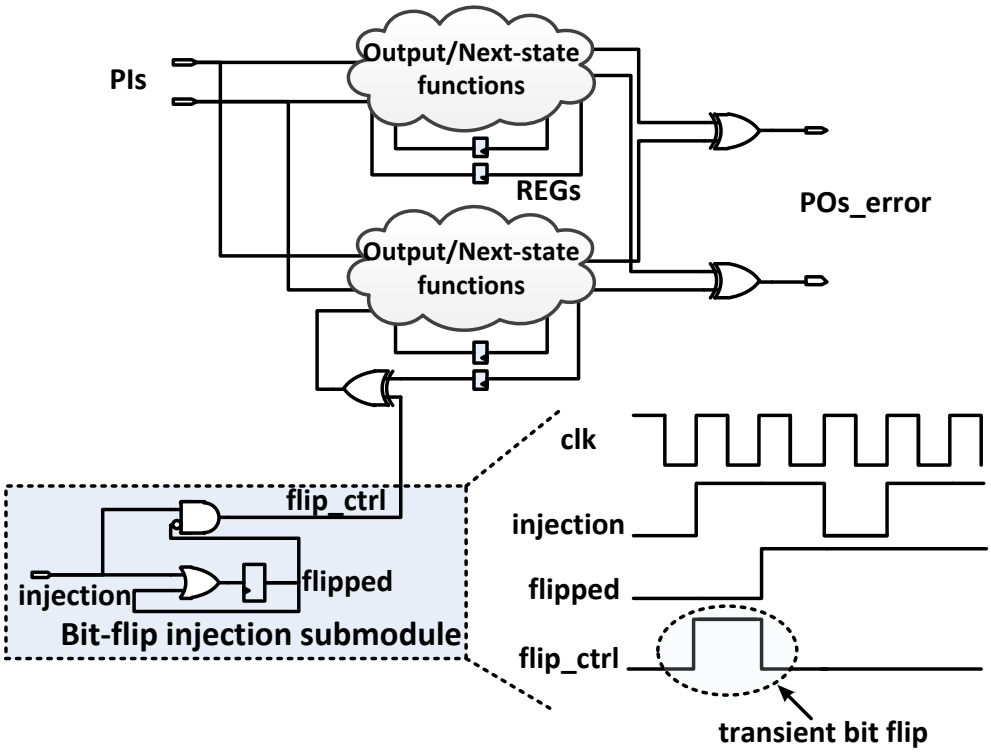


Figure 4.6: Construction of the error checking module

For fault simulation purpose, only one additional primary input and an *XOR* gate are necessary. The user can inject a single bit-flip in the register during the RTL simulation. However, for SEC purpose, only one *XOR* gate is not enough. SEC is a formal technique and it assumes that the value of any primary input will always have two possibilities: low or high. To model a transient soft error in SEC, we integrate a new *bit-flip injection submodule* into the faulty design, as shown in Figure 4.6. Assume the *flipped* flag signal is initialized with low value, and after the input *injection* becomes high, the *flip_ctrl* output will be high for only one clock cycle and then always stay low, no matter how *injection* changes.

After the integration of the bit-flip injection submodule, the entire error checking module is sent to an SEC tool for design simplification [136]. By merging the equivalent registers and removing the redundant logic, a compacted error checking module is obtained. The compaction potential of this method is significant for larger designs, because they manifest much locality of error influence. As shown later in our experiments, on average this compaction can reduce the number of REG modeling variables by more than 30%.

Time-multiplexed PIs Partitioning (TPP)

Except for reducing the number of REG variables, another technique, called *time-multiplexed PIs partitioning*, is proposed to handle the design with large number of PIs. This technique divides the entire huge state space into small partitions, and then analyzes them sequentially. Generally the output and next-state functions of each individual PO/REG depend only on limited number of PIs rather than all of them, especially for a design with large number of

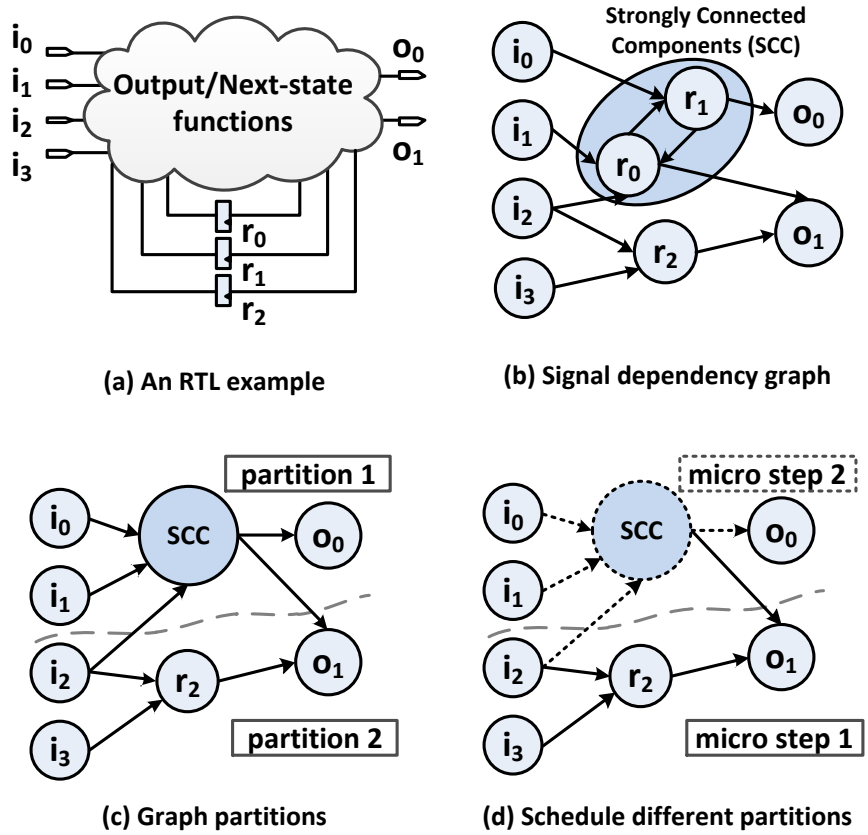


Figure 4.7: Time-multiplexed PIs partitioning

PIs. Therefore, one *time step* of the original design can be divided and modeled by several *micro-steps*. In each micro-step, only a fraction of PIs, POs and REGs values will be assigned, calculated and updated. That is, we replace a set of PI assignments working in *parallel* with a time-multiplexed version working in *sequence*. It explores the inherent structural decomposition of the RTL designs, and alleviates the state explosion problem. In different micro-steps, with *different* input probabilistic distributions the *same* PI modeling variables can be reused for different PO/REG value assignments. The functionalities of the original and partitioned designs are equivalent, but the computation effort for each partition can be much smaller than directly analyzing the entire state space.

The key challenge here is how to partition the design. The interactions among different partitions should be minimized, while the signal dependencies must be preserved to guarantee the functional correctness. To achieve this, the following work flow is proposed:

1. Construction of the PI/PO/REG *Signal Dependency Graph* (SDG) G;
2. Condensation of G into an associated *directed acyclic graph* (DAG) G_{scc};
3. Min-cut partitioning of G_{scc};
4. Scheduling of the partitions into different micro-steps.

Taking Figure 4.7(a) as an example, there are four PIs i_0, i_1, i_2, i_3 in this design. First the SDG G is established as in Figure 4.7(b). In this SDG the RTL signals correspond to the graph nodes, and the edges represents the signal dependencies in the output and next-state

functions.

Due to the existence of feedback loops among REG signals, G could be a cyclic graph. Partitioning of such graph can introduce mutual dependent signals, making their treatments at the partition boundaries very complicated. Therefore, we extract the *Strongly Connected Components* (SCCs), and then build G_{scc} by collapsing together all the nodes of each SCC. In this G_{scc} signal dependencies are unidirectional and the mutual dependencies across different partitions are removed, as in Figure 4.7(c). To find partitions with minimum interactions, G_{scc} is min-cut partitioned [137]. As the maximum number of modeling variables in all micro-steps determine the largest space size of the partitioned design, the number of PI variables in different partitions should be balanced. Finally the partitions are scheduled in a way that the signal dependencies are preserved as much as possible. This can reduce the number of additional modeling variables, which are necessary to store intermediate values. For this example in Figure 4.7(d), partition 2 is scheduled in micro-step 1 (before partition 1). As the value of o_1 depends on the value of r_0 in current time step, o_1 needs to be calculated before the value of r_0 is updated in micro-step 2. If we use s and s' to represent the signal value in current and next time step respectively, the scheduling of these micro steps and the corresponding signal value update are as follows:

- *Micro step 1*: $r_2' = f_{r_2}(i_2, i_3)$, $o_1 = f_{o_1}(r_0, r_2)$;
- *Micro step 2*: $r_0' = f_{r_0}(i_1, i_2, r_1)$, $r_1' = f_{r_1}(i_0, r_0)$, $o_0 = f_{o_0}(r_1)$.

Note that $f_s(\cdot)$ represents the Boolean output function of PO signal s or next-state function of REG signal s. It can be seen that the two subsequent micro steps perform exactly the same operations as one original time step with simultaneous update of all signal values. That is to say, the partitioned time-multiplexed design is *functionally equivalent* to the original design.

To see the advantage of this PI partitioning, assume each PI corresponds to one variable in model checking, then there are $2^4 = 16$ states in the original PI space. For this partitioning and scheduling scheme, at most three variables v_0, v_1, v_2 are necessary to model the PI variables: micro-step 1 ($i_3 = v_0, i_2 = v_2$) and then micro-step 2 ($i_0 = v_0, i_1 = v_1, i_2 = v_2$). Note that v_0, v_1 can have different values across different micro-steps, while v_2 keeps the same value, because i_2 is shared in partition 1 and 2. By reusing the variables across different micro-steps, the number of modeling variables is reduced from the previous 4 to current 3. For this small running example the reduction is not significant, but for larger designs there can be more benefits due to the partial PI dependency, which is around 50% variable reduction as illustrated in Section 4.6.

Further Discussion on Scalability

The above scalability improving techniques either remove the modeling redundancy, or explore the structural decomposition of the state space. Therefore, they maintain the exhaustive feature of formal model checking to explore the entire search space. If accuracy can be sacrificed to further increase the scalability, statistical model checking can also be adopted [138]. It is a scalable verification technique that provides statistical guarantees of the verification results. The basic idea is to perform the verification of the DTMCs by sampling the executing paths, therefore it can obtain the approximate probabilities within tolerable bounds of error. Unlike the exhaustive model checking with numerical analysis on DTMCs, it avoids the construction

of the entire state space, hence it is particularly suitable for very large models where normal model checking is infeasible.

4.4 Word-level Error Analysis in RTL Data Paths

The key idea of data path error analysis is to derive EPP rules for different RTL operations with an accurate analytical method of Error Propagation Probability (EPP). This feature makes our approach distinctive from traditional statistical fault injection method. Moreover, since the proposed analytical method is performed at register-level (word-level), the complexity of our approach is considerably reduced compared with bit-level techniques.

4.4.1 Basic Idea

The basic approach of our proposal is derived from the preliminary work about gate-level SER estimation [139]. This work presents an analytical approach of traversing structural paths from struck site to primary outputs. It uses propagation rules of different gate types for on-path signals and signal probabilities (SP) for off-path signals. Thus in only one pass, the error probabilities of all the on-path signals will be computed.

Taking advantage of the similarity, it is possible to compute and estimate the EPP, i.e. the vulnerability, of each RTL building block, rather than gate mentioned in previous work. That is to say, we raise the abstraction level from gate level to RT level. Take Figure 4.8 as an example, particle strike may cause bit errors in the primary input **B**, then with successive operations along the paths, these error may propagate to primary outputs **F** and $\mathbf{C}_{out}$. By calculating and obtaining the vulnerability of each register, we can average all of them to get the vulnerability value for the current RTL block. In similar way, EPP (vulnerability) values for different RTL parts in the system could be computed. As long as the occurrence rate of soft error in i^{th} register $R_{SE}(i)$ is known, we could estimate the SER of the whole system as

$$SER_{total} = \sum SER(i) = \sum R_{SE}(i) \cdot EPP(i)$$

where the parameter $R_{SE}(i)$ depends on the energy of particle, type and size of gate, active area and device characteristics. In the estimation phase, this reliability parameter should be provided by the cell library on which the system is planed to implement later.

Thus it is possible to distinguish the most vulnerable parts in the system and apply appropriate soft error mitigation techniques taking into consideration the reliability metric.

4.4.2 Bit-level *vs* Register-level

From the hardware perspective, the basic storage element at RTL is register. Between different registers are combinational logic circuits that finish specific operations. For synchronous circuits, one common clock signal synchronizes all the output from combinational parts and updates the content in each register. When we consider soft errors at RTL level, they manifest themselves as bit-flips, so the consideration of EPP at RTL level could also have two directions: bit-level EPP and register-level EPP. The former has the advantage of better handling of bit-flips on basic operations, especially for operations with bit manipulation such as shift

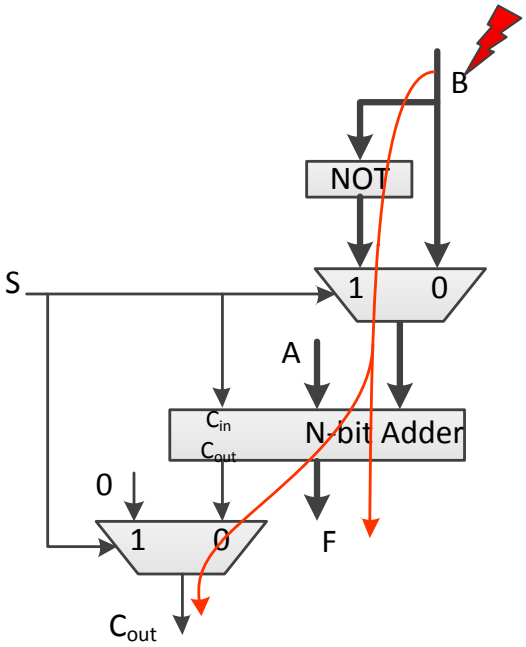


Figure 4.8: RTL-level error propagation

and comparison. However, with the wordsize for each register and the number of total registers scaling up, the computational complexity of bit-level EPP increase dramatically, which unfortunately deviate from our initial intention of RTL soft error modeling. As an alternative, register-level EPP method considers register with multiple bits as a whole and therefore simplify the modeling of error propagation for different operations. Based on some reasonable assumptions, it can still guarantee satisfactory accuracy, just as what will be demonstrated later.

Therefore, our proposed approach is actually register-level EPP calculation among different RTL building blocks. To facilitate later analytical EPP computation and analysis, except for the regular register properties like word size and bit values, two additional properties need to defined:

- Definition: *Error Probability* $P_e(X)$
  the probability that the value of register X is erroneous. We assume that any one bit error in the register will cause the whole register to be faulty;
- Definition: *Signal Probability* $SP(X)$
  the probability of each bit in the register X to be 1. It can be formulated as following equation:

$$SP(X) \equiv \frac{1}{N} \sum_{i=0}^{N-1} SP(X[i])$$

where N is the word size and $SP(X[i])$ is the probability of i^{th} bit of register X to be 1;

4.4.3 Register-level EPP Rules

A basic action in register transfer methodology is register transfer operation, which can be described as following notation for a register transfer operation [125]:

$$r_{dest} \leftarrow f(r_{src1}, r_{src1}, \cdots r_{srcn})$$

where r_{dest} is the destination register and on the right side (r_{src1}, r_{src1} and r_{srcn}) are the source registers. After previous definition and assignment of necessary properties to each register, error propagation rules should be formulated for register transfer operations.

As complex operations can always be decomposed into simple and basic operations, here only EPP rules for the most common, basic and furthermore hardware synthesizable operations are considered and formulated. Generally speaking, these operations could be categorized into three groups:

- Arithmetic
  Here only *Addition* is considered as a representative of the whole category, e.g. $Z = A+B$;
- Bitwise Logic
  The most elementary operations are *AND*, *OR* and *NOT*, e.g. $Z = A\&B$, $Z = A|B$, $Z = \sim A$;
- Shift
  For simplicity, only *Logic shift* is considered here, e.g. $Z = A \ll k$ and $Z = A \gg k$, where k is the number of shifted bits and is considered as a constant value;

Register-level EPP Assumptions

Before using the analytical approaches to estimate EPP, the employed assumptions are explained and listed as following:

- *Assumption 1*: The two input registers of basic operations listed above are independently faulty, which means their error probabilities are not related;
- *Assumption 2*: Each bit in the same register is independent, and its signal probability is the same as the overall SP of this register. Besides, each bit's error probability is not related to others and assumed to be $1/N$ of the register's error probability (N is the word size of this register).

For the investigation of data paths, *Assumption 2* is reasonable because for the data register, its content could be roughly considered to be random. Although logic shift operations would generate several continuous 0 bits, which may make this assumption debatable, it is just one source of inaccuracy and as shown later in the experimental part, the inaccuracy resulting from this is in tolerable range. It may also be argued that for *Assumption 1*, convergent paths in the data flow would cause some dependency between several input registers, but just as mentioned before and validated later, the simplicity of analysis and computation is worth of some acceptable inaccuracy.

EPP Rules for Basic Operations

Based on the assumption before, EPP rules for basic operations are derived. The error probability of output register can be generally described as

$$P_e(Reg_o) = P_e(Reg_{in1} \cup Reg_{in2} \cup \cdots \cup Reg_{inN}) - P_{mask}$$

where N is the number of input ports and P_{mask} is the probability that the errors at the inputs are masked and cannot propagate and appear at the output.

Now it is ready to formulate the EPP rules of basic operations:

- Arithmetic
  For arithmetic operations like $Z = A + B$, as they generate accurate value, EPP is roughly considered to be 1. Some extreme cases may bring about masking phenomenon, such as $Z = A + (-A)$ and $Z = A * 0$, but for simplicity these rare cases are ignored, that is to say, $P_{mask} \approx 0$. Therefore, EPP rules for $Z = A + B$ is

$$\begin{aligned} P_e(Z) &= P_e(A \cup B) - P_{mask} \\ &\approx P_e(A) + P_e(B) - P_e(A \cap B) \\ &\approx P_e(A) + P_e(B) - P_e(A) \cdot P_e(B) \end{aligned}$$

  where the second approximation is from *Assumption 1.*

- Bitwise Logic
  For *NOT* operation $Z = \sim A$, obviously the EPP is 1, that is to say, error in the input register will definitely propagate to output register, therefore

$$P_e(Z) = P_e(A);$$

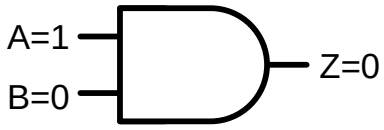


Figure 4.9: *AND* operation EPP

  For logic *AND* operation $Z = A\&B$, the EPP calculation is a little complicated and a method similar to deductive fault simulation [140] is used. Here we take one case $A = 1, B = 0$ as an example in Figure 4.9. The probability of this case is $SP(A) \cdot (1 - SP(B))$, and only when B is faulty and A is fault-free, the output Z is faulty, so the error probability of output register can be derived as

$$\begin{aligned} P_e^{1,0}(Z) &= P_e(\bar{A} \cap B) \\ &= P_e(B) - P_e(A \cap B) \\ &\approx P_e(B) - P_e(A) \cdot P_e(B) \end{aligned}$$

  In the similar way, we could get the $P_e(Z)$ for the other three cases $A = 0, B = 0$; $A = 0, B = 1$ and $A = 1, B = 1$. Finally, the EPP rule for *AND* operation could be

obtained as

$$
\begin{aligned}
&P_e(Z) \approx \\
&[P_e(A) + P_e(B) - P_e(A) \cdot P_e(B)] \cdot SP(A) \cdot SP(B) + \\
&[(P_e(B) - P_e(A) \cdot P_e(B)] \cdot SP(A) \cdot (1 - SP(B)) + \\
&[P_e(A) - P_e(A) \cdot P_e(B)] \cdot SP(B) \cdot (1 - SP(A)) + \\
&[P_e(A) \cdot P_e(B)] \cdot (1 - SP(B)) \cdot (1 - SP(A))
\end{aligned}
$$

For logic *OR* operation, the EPP rule is derived similarly as

$$
\begin{aligned}
&P_e(Z) \approx \\
&[(P_e(B) - P_e(A) \cdot P_e(B)] \cdot SP(B) \cdot (1 - SP(A)) + \\
&[P_e(A) - P_e(A) \cdot P_e(B)] \cdot SP(A) \cdot (1 - SP(B)) + \\
&[P_e(A) \cdot P_e(B)] \cdot SP(A) \cdot SP(B) + \\
&[P_e(A) + P_e(B) - P_e(A) \cdot P_e(B)] \cdot \\
&(1 - SP(B)) \cdot (1 - SP(A))
\end{aligned}
$$

Up to now, we already have the EPP rules for elementary logic operations *AND*, *OR* and *NOT*. As mentioned before, more complicated logic operations could be decomposed into combination of elementary logic operations, it is easy to use the EPP rules available now to obtain EPP rules for *NAND*, *NOR*, *XOR* and *XNOR* operations.

- Shift
  As illustrated in Figure 4.10, the number of shifted bits k is a constant and assumed to be

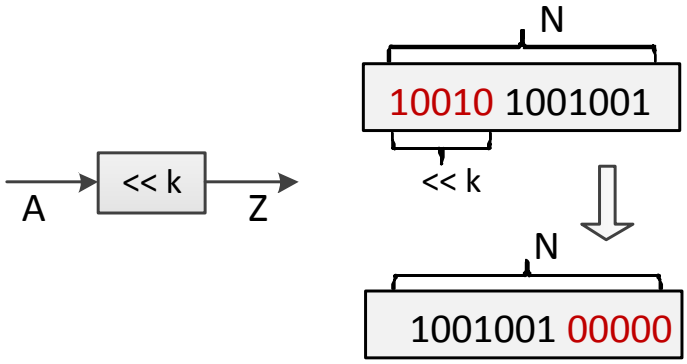


Figure 4.10: *Shift* operation EPP

fault-free. Using the general formula mentioned above, for $Z = A \ll k$ and $Z = A \gg k$, we have

$$P_e(Z) = P_e(A) - P_{mask};$$

To calculate the error probability of output register Z, the masking probability need to

be obtained first for different cases:

$$\text{1 bit error in A: } P^1_{mask} = \frac{k}{N};$$
$$\text{2 bit error in A: } P^2_{mask} = \frac{C^2_k}{C^2_N} = \frac{k(k-1)}{N(N-1)};$$
$$\vdots$$
$$m \text{ bit error in A: } P^m_{mask} = \frac{C^m_k}{C^m_N} (m \leq k)$$

where C^m_k is m-combination of set k. The computation of $P_e(Z)$ also requires the probability for each case above, which, however, are not available for us. We only know the probability of erroneous Z, rather than probability of different number of bit errors. For simplicity and raw estimation, we only consider 1 and 2 bit error cases and more bit error cases are ignored. Therefore, using following equations:

$$P^1_{mask} + P^2_{mask} = 1$$
$$(P^1_{mask})^2 = P^2_{mask}$$

We set 0.6 and 0.4 as raw probabilities for 1 and 2 bit error in input register A and finally get the EPP rules for *Shift* operation:

$$Pe(Z) = Pe(A) \cdot [1 - (0.6 \cdot \frac{k}{N} + 0.4 \cdot \frac{C^2_k}{C^2_N})]$$

4.4.4 Overall EPP Estimation

After the formulation of EPP rules for basic operations, the overall EPP of an RTL block can be estimated as following, where high-level Data Flow Graph (DFG) is used as the input:

1. Traverse the whole DFG to obtain the register set of concern, assume its number is N;
2. For each register in this set, the following steps are performed:
   - Initialization: for the register Reg_i at error site, $P_e(Reg_i) = 1$ and all other registers $P_e = 0$;
   - Traverse from primary inputs of DFG and apply EPP rules for each operation to obtain the output register's error probability until it reaches the primary outputs. The P_e of register on the reachable paths from error site Reg_i to primary outputs would be updated automatically and P_e of other registers are still 0;
   - Assume there are M reachable output registers Reg_O from Reg_i, and as error in any output register would be considered as the overall outputs are wrong, we use following equations to calculate EPP of Reg_i

$$EPP(Reg_i) = 1 - \prod_{i=1}^{M} [1 - P_e(Reg_O(i))];$$

3. After all registers are repeated on step 2, the EPP of the whole RTL block is computed as

$$EPP_{overall} = \frac{1}{N} \sum_{i=1}^{N} EPP(Reg_i);$$

4.5 Vulnerability Analysis at Behavioral Level

To evaluate the variable and operation vulnerabilities to soft error, several important factors should be taken into account, including lifetime, functional dependencies, weight in conditional branches and error masking effects. In this work, we propose a state-based vulnerability evaluation technique to analyze both data and control errors, which inherently takes into account all the above mentioned factors, and can efficiently handle the sophisticated branch and loop structures in the control flows of the behavioral designs.

The vulnerability evaluation in this work is based on single error model and performed on the CDFG of the behavioral design, which is a directed graph consisting of two level structures: control flow graph and Basic Blocks (BB). Each BB includes a sequence of operations to perform the computation i.e. DFG, and the last one is associated with a branch condition to indicate the control dependencies between BBs, as shown in Figure 4.11(a). Without loss of generality, we assume that the CDFG has a single entry BB and a single exit BB. The variable/operation vulnerability evaluation in this CDFG consists of three main parts: random error occurrence modeling, error propagation modeling and the vulnerability evaluation.

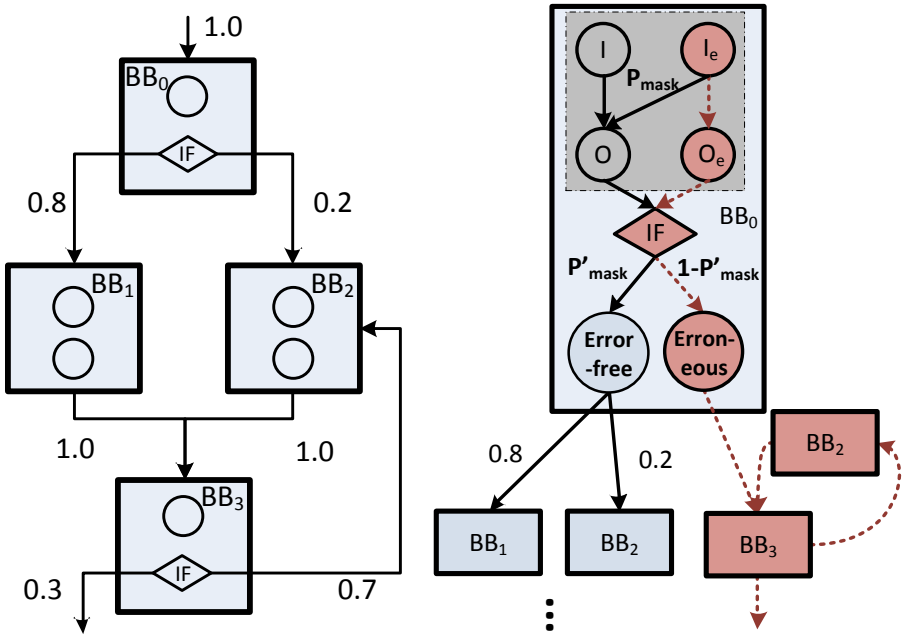
4.5.1 Random Error Occurrence Modeling

The random property of soft errors means that they can occur at any possible moment of the program execution starting from the function entry to exit. Taking different branch probabilities and error masking effects into consideration, two subtle scenarios make the vulnerability analysis a challenging task: errors occur and propagate along the branch with low (high) execution probability, while the error masking probability is also very low (high). In these scenarios, large amount of paths have to be investigated to check the contribution of the errors, which propagate on these paths and finally arrive at function exit.

To efficiently handle this issue without sophisticated path enumeration and time-consuming fault injection, we model the occurrence of soft error as a nondeterministic phenomenon by Markov Decision Process (MDP) [127], which can be viewed as a variant of Markov chain that permits both probabilistic and nondeterministic choices. We use one state variable *phase* to indicate whether soft errors occurred or not. For each state with $phase = 0$ (error has not occurred yet), the system nondeterministically makes transitions to either $phase = 0$ or $phase = 1$ (error occurs). Following $phase = 1$ are the states with $phase = 2$ (error propagates).

4.5.2 Error Propagation Modeling

The investigation of error propagation in a software system has been studied in [141, 142]. While the work in [141] considered only the control errors, the technique in [142] did not differentiate data and control errors while propagating them in the software components. In this work, we should handle the two types of errors separately, as they have different error propagation and masking mechanisms. The basic idea on modeling data error propagation is to construct a macro-state with error-free and erroneous transitions for each operation in the BB, as shown in Figure 4.11(b). For one operation with the input I, output O and function $O = f(I)$, the macro-state consists of four states:



(a) CDFG with branch probabilities (b) Error propagation model

Figure 4.11: Control data flow graph example and our adopted error propagation model

- I: all the inputs of this operation are error-free;
- I_e: at least one of the inputs of this operation is erroneous;
- O: the output of this operation is error-free;
- O_e: the output of this operation is erroneous;

To model the error masking effects, one special transition from I_e to O is added with an associated masking probability P_{mask} (e.g. one *AND* operation can theoretically mask half of the input errors with $P_{mask} = 0.5$ and generate the correct output). Note that for different operation types P_{mask} can be different, and in this work we adopt the masking probability values from [123].

When the errors propagate to the conditional *IF* branch (i.e. control errors) as shown in Figure 4.11(b), we can use a conservative assumption [141] that the values of variables calculated in wrong branches are erroneous. The rationale behind this assumption is that: i) if the same operation with same operands is carried out no matter whether the condition is true or not, it can be moved just outside the conditional statements by the common *redundancy elimination* compiler optimization [143]; ii) the behavioral variables always have wide width (e.g. 32-bits integers) and the probability that variables from different branches have the same values is rather low. Therefore, due to the control error (with probability $1 - P'_{mask}$) in *IF* condition of BB_0 in Figure 4.11(b), the state makes transition to BB_3 directly and the variables calculated in BB_1 and BB_2 are set to be erroneous.

Note that this propagation modeling is not limited to be at operation level, and it is also possible to build the macro-states of error propagation and masking for each BB or even function. Therefore with different input error probabilities we are able to calculate the corresponding output error probabilities for each macro-state, and propagate errors across

macro-states, i.e. support hierarchical modeling of the error propagation to reduce the overall complexity.

4.5.3 Vulnerability Evaluation

In the last step, the error occurrence and error propagation model need to be combined. In this work we exploit the parallel composition mechanism, which means for one MDP composed with several parallel submodules, the global states of the combined MDP are the interleaved states of each submodule [127]. In this way, the occurrence of soft error is interleaved with each possible state in the error propagation submodule. This means before soft error occurs, the combined MDP just follows the normal execution and every variable is error-free, and after soft error nondeterministically occurs, the combined MDP proceeds with the error propagation model. The propagation terminates when the MDP reaches one sink state representing the function exit and the error probabilities of the function output will be evaluated at this sink state.

The probabilistic model checking tool PRISM [129] is employed in our work to automatically reason about the observed error probability at function exit over all possible resolutions of the nondeterminism. The obtained error probabilities are conservative estimation of the behavioral vulnerabilities, which are appropriate for the variable and operation ranking as a guideline of error mitigation.

4.6 Experimental Results

To evaluate the proposed techniques for register vulnerability analysis, we carried out the experiments on several representative RTL control paths and data paths. In addition, a case study on an RTL cache controller is performed to demonstrate the non-uniform register vulnerabilities, and the quantification of workload dependency. The experiments were performed on a workstation with Intel Xeon E5540 2.53GHz and 16GB RAM.

4.6.1 Control Path Evaluation Results

Six RTL control path benchmarks are used, including the control FSMs from the data management unit of OpenSPARC [144], instruction/data cache (IC/DC) controller, and program counter (PC) generator of the OpenRISC 1200 (OR1200) processor [109].

The vulnerability evaluation flow is illustrated in Figure 4.12. First the RTL description is processed for data type reduction, especially for the signals with large bitwidth. Then the design is partitioned and modeled as a PI-multiplexed version. For each register to be evaluated, the error checking module is constructed as in Figure 4.6 and simplified by SEC. The corresponding DTMC models are generated and then using the prepared property file, the PRISM model checker generates the register vulnerability. Note that the three techniques are actually orthogonal to each other, and they can be applied in any user specified order. According to the previous analysis in Section 4.3.4, DTR and TPP techniques are independent of the error sites (i.e. impaired registers), therefore the two processing steps can be performed only once when the vulnerabilities of multiple registers need to be evaluated.

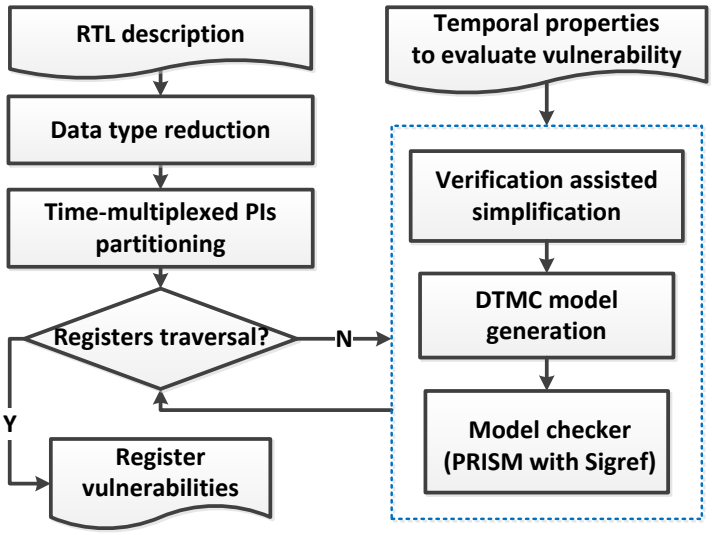


Figure 4.12: Register vulnerability evaluation flow in control paths

Table 4.2: Benchmark characteristics and the runtime for register vulnerability evaluation

Benchmarks	PIs	REGs	Model Construction Time (sec.)	Model Checking Time (sec.)
rmu_rrm_etsbfsm	5	8	0.7	0.3
ilu_eil_xfrfsm	15	6	0.4	0.9
clu_crm_pktctlfsm	15	14	85.4	5.1
or1200_ic_fsm	40	42	230.5	1.3
or1200_dc_fsm	102	48	1927.9	0.1
or1200_genpc	241	95	2330.2	0.1

It is worth to mention that actually the proposed technique for error evaluation is not limited to RTL description. In principle, if a gate-level design is modeled as a state transition system, the proposed method can also be applied to a pure gate-level description. However, in terms of scalability it is not efficient to use formal model checking technique for gate-level design, because all the word-level information are lost and the signals are flattened bit by bit, which will deteriorate the scalability of applying formal methods. Therefore, we start from RTL, and make full use of the high level abstraction efficiency to reduce the model size. In other words, we exploit the power of formal model checking techniques for RTL designs in which the disadvantage of such methods, i.e. scalability, can be hidden by doing the analysis at a higher level of abstraction which allow us to reduce the state space. In summary, this technique is more appropriate for error evaluation in an RTL description rather than a pure gate-level netlist.

Table 4.2 shows the characteristics of the employed benchmarks as well as the runtime for evaluating the register vulnerabilities. For the first three small benchmarks, direct modeling and analysis of the register vulnerabilities is possible, therefore the DTR, VAS and TPP techniques are not adopted. For the remaining benchmarks, due to the large size of PIs and REGs all the abstraction and simplification techniques are applied, then the PMC runtime are reported based on the reduced design.

Table 4.3: PMC variables reduction and corresponding runtime by applying three scalability improvement techniques

Benchmarks	PI Variables				REG Variables				Runtime (sec.)				
	Ori.	DTR	TPP	DTR +TPP	Ori. [1]	DTR	VAS	DTR +VAS	Ori.	DTR	DTR +TPP	DTR +VAS	DTR+ VAS+TPP
or1200_ic_fsm	40	10	21	**9**	84	24	72	**21**	TO[1]	719.7	541.4	306.4	231.8
or1200_dc_fsm	102	25	75	**11**	96	42	33	**12**	TO	TO	TO	TO	1928.0
or1200_genpc	241	37	58	**19**	190	26	154	**18**	TO	TO	TO	TO	2330.3
Avg. reduction	-	78%	50%	**86%**	-	71%	33%	**84%**	-	-	-	-	-

[1] As shown in Figure 4.2, the number of REGs for error checking includes both fault-free and faulty designs.
[2] Time Out (TO): in these experiments, the time limit for model checker was set to be 3 hours.

The model construction time includes the time taken for preprocessing the RTL designs as well as the bisimulation time. While the preprocessing steps (i.e. DTR, VAS and TPP) need only several seconds, the bisimulation process dominates the model construction time, because it tries to compact the equivalent states as much as possible. With the reduced state space, the model checking time is very short.

Table 4.3 shows the scalability improvement of our proposed techniques, when they are applied separately or all-together to the three larger benchmarks. Note that VAS technique only aims to reduce REG variables, while TPP only for PI variables. As the state space size increases exponentially with the number of PI/REG variables, direct model checking without abstraction and simplification is not feasible. Therefore, the numbers of PI and REG modeling variables in DTMCs, instead of the runtime, are reported here. The numbers of REG variables in VAS technique are averaged over all possible error sites, and the PI variables in TPP include the PIs in individual micro-step as well as the shared PIs across multiple micro-steps, as illustrated in Figure 4.7.

The detailed comparison of all three techniques regarding relative variable reduction is shown in Figure 4.13. The DTR technique leverages the efficient RTL abstraction, therefore it can reduce the number of modeling variables on average by 78% and 71% for PIs and REGs. In comparison, applying VAS or TPP separately is able to reduce the number of REG or PI variables by 33% or 50%. If all three techniques are applied together, the total reduction of the PI and REG variables can reach 86% and 84%, respectively. As the overall state space size increases exponentially with the number of PI/REGs variables, such amount of reduction has tremendous impact on the PMC runtime when the three techniques are applied separately. As expected, there are many timeout scenarios without proper state space reduction, which directly impact the feasibility of probabilistic model checking for register vulnerability evaluation.

To investigate the efficiency of proposed scalability improving techniques, we use two additional larger benchmarks: 64-bit version of the controller *or1200_genpc_64* with double bitwidth signals, and duplication of the controllers *or1200_genpc_dup* with double PIs and REGs but the same number of POs. The corresponding runtime and comparison with original design without PMC variable reduction are illustrated in Figure 4.14. Note that for the original benchmarks without simplification, the exponential runtime is extrapolated based on the available actual PMC runtime of the smaller benchmarks, for which the model checking was able to successfully terminate. From this investigation we can see that although the PMC

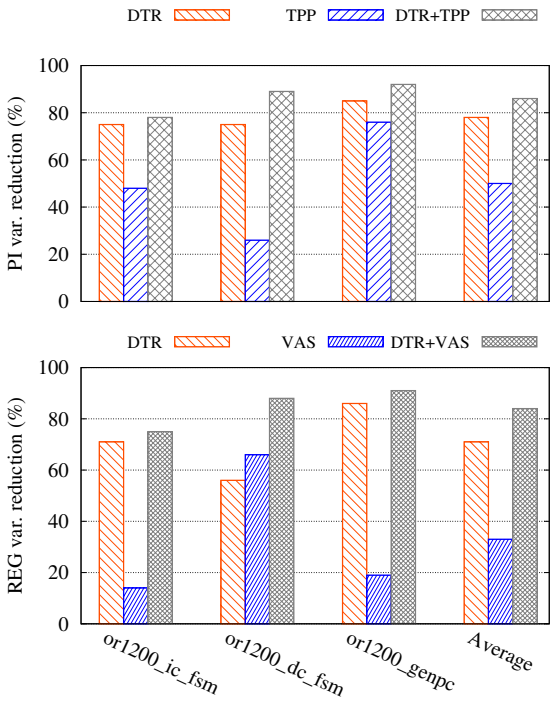


Figure 4.13: Relative reduction of modeling variables with three different techniques

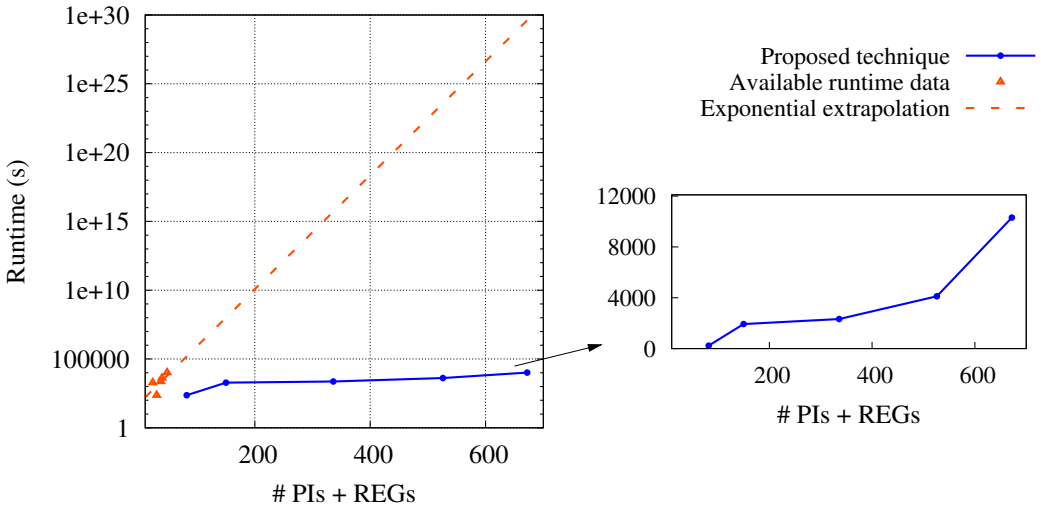
runtime increases exponentially with the number of modeling variables, with the proposed variable reduction techniques, this trend is significantly alleviated and the PMC-based vulnerability evaluation can be well applied to handle complex control modules. Considering the fact that the vulnerability evaluation of different control units could be evaluated independently and be fully parallelized, the runtime of applying the proposed technique to other controller units can be estimated with this largest block. As this work focuses on the control path analysis, regarding the entire processor, there are another set of orthogonal techniques like [126] for analyzing data path blocks.

4.6.2 Data Path Evaluation Results

For data path EPP evaluation, we use 8 DFG examples for evaluation and these benchmarks can be categorized into two groups:

- The first 4 DFGs (benchmark 1∼4) are manually constructed including all the mentioned basic operations and the usual convergent paths. To evaluate the inaccuracy caused by shift operation and convergent path, special cases that only contain convergence or shift operations are designed;
- The other 4 DFGs (benchmark 5∼8) are from algorithm DFGs in the literature [145–147], which are Secure Hash Algorithm, ADPCM encoder and FIR filter.

An overview of these 8 DFGs are shown in Table 4.4.



(a) Runtime comparison with the exponential case without PMC variable reduction

Benchmarks	Original		Reduced Vars		Runtime
	PIs	REGs	PIs	REGs	(sec.)
or1200_ic_fsm	40	42	9	21	231.8
or1200_dc_fsm	102	48	11	12	1928.0
or1200_genpc	241	95	19	18	2330.3
or1200_genpc_64	339	187	19	20	4117.5
or1200_genpc_dup	482	190	24	24	10311.8

(b) Benchmark statistics and runtime

Figure 4.14: Scalability investigation with large benchmarks

Table 4.4: Overview of the benchmark DFGs

Bench	Operations	Registers	Input registers	Output registers	Additional notes
DFG1	8	12	4	2	All basic operations included with convergent path
DFG2	8	11	3	2	All basic operations, stronger convergence
DFG3	8	11	3	2	Without SHIFT operation but with convergence
DFG4	6	10	4	2	With SHIFT operation but without convergence
DFG5	13	17	4	2	SHA algorithm DFG
DFG6	13	17	4	2	SHA algorithm DFG without SHIFT operation
DFG7	9	13	4	2	ADPCM encoder algorithm DFG
DFG8	9	10	1	1	FIR filter DFG

Validation Flow of EPP Rules

To verify the accuracy and speed advantage of the proposed EPP rules, a validation flow is conceived and illustrated in Figure 4.15.

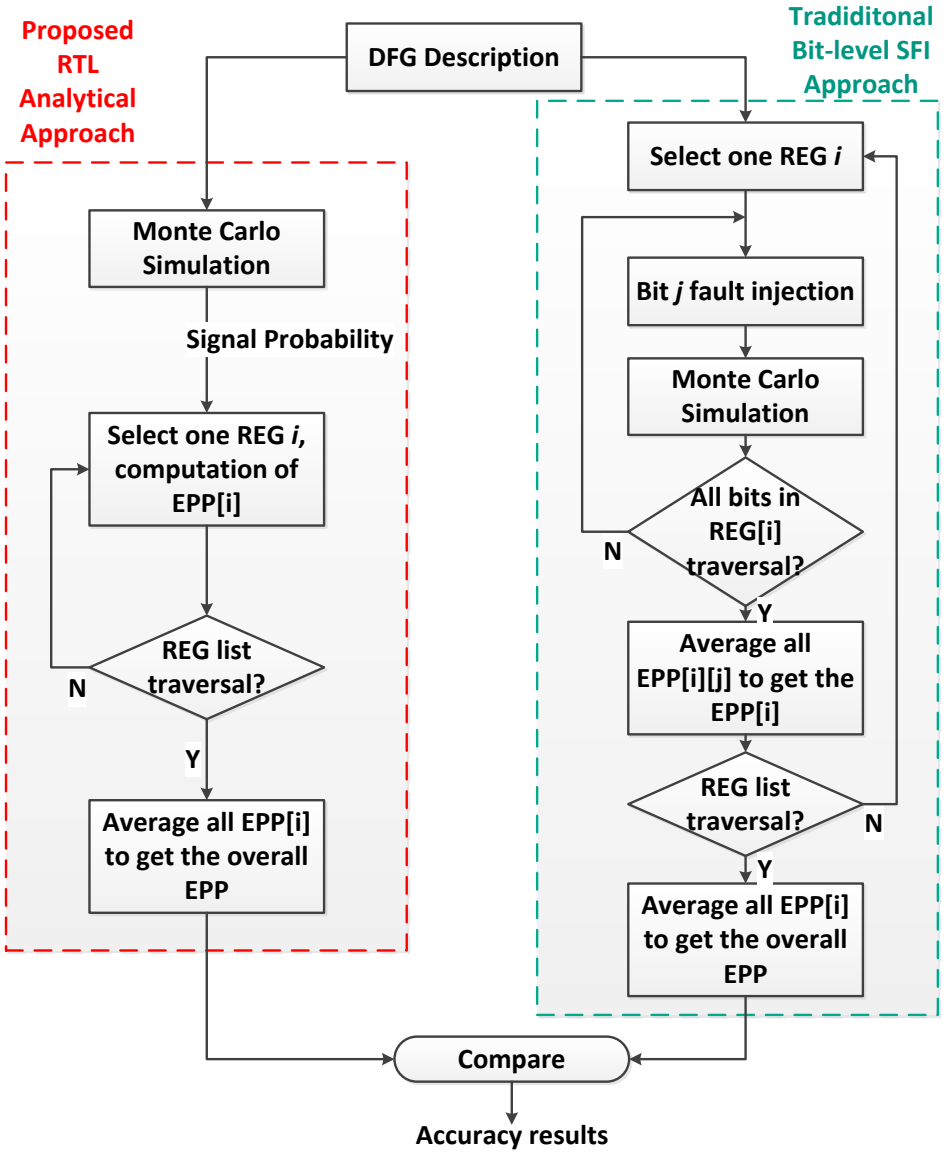


Figure 4.15: Validation Flow of the EPP rules

The input of validation flow is the DFG description. After the initial traverse of the whole DFG, registers of concern to be investigated are obtained. Then there are two different branches for verification:

- For each register of concern, using Statistical Fault Injection (SFI) based on Monte-Carlo simulation, fault is injected into each bit and simulated, primary output are compared with expected values in fault-free simulation and then EPP for one bit is obtained. Average of these bit EPPs gives EPP of the whole register. After traversing all registers of concern, average of them gives fault-simulation EPP of the overall DFG;

- First fault-free Monte-Carlo simulation generates SP for each register, and then with fault occurring in each whole register rather than specific bit, propagation rules are applied to each operations in the DFG and EPP of the impaired register is obtained. After traversing all registers of concern, average of them gives analytical EPP of the overall DFG.

Comparing the EPP value from two different branches give us the accuracy of our analytical approach and runtime comparison could tell the speedup ratio of our method.

Comparison Results

With the SFI simulation of 99.9% confidence level, the EPP results and corresponding comparison are shown in Table 4.5, Figure 4.16 and Figure 4.17.

Table 4.5: Comparison of proposed analytical EPP approach with traditional SFI simulation regarding accuracy and runtime (sec.)

Bench	Analytical value	SFI value	Absolute difference	Relative difference	Analytical runtime	SFI runtime	Speedup (w/o SP)	SP runtime	Speedup (with SP)
DFG1	0.624	0.598	0.043	7.3%	5.0E-06	208.0	4.1E+07	5.5	37.6
DFG2	0.602	0.542	0.060	11.0%	8.0E-06	196.0	2.4E+07	4.8	40.7
DFG3	0.430	0.440	0.010	2.2%	6.0E-06	139.8	2.3E+07	4.7	29.3
DFG4	0.715	0.594	0.122	20.4%	4.0E-06	111.9	2.8E+07	0.4	226.7
DFG5	0.777	0.680	0.096	14.1%	1.0E-05	499.6	5.0E+07	7.2	68.9
DFG6	0.741	0.662	0.079	11.9%	1.1E-05	403.6	3.6E+07	7.4	54.4
DFG7	0.559	0.555	0.003	0.6%	9.0E-06	267.2	2.9E+07	6.9	38.7
DFG8	0.999	0.987	0.012	1.2%	4.0E-06	615.4	1.5E+08	4.1	37.6
Average	-	-	0.053	8.6%	-	-	4.8E+07	-	80.7

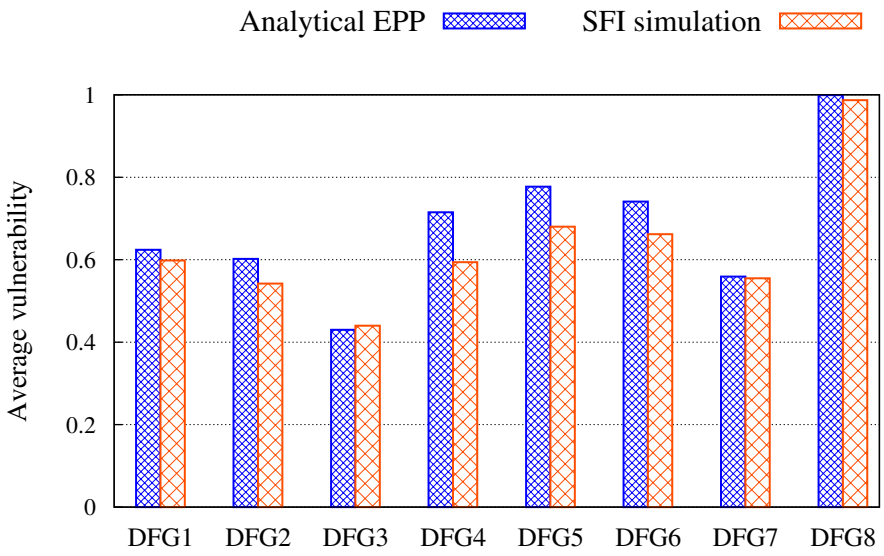


Figure 4.16: Accuracy comparison of different DFG benchmarks

The experimental results shows that for 8 benchmarks, compared with traditional SFI simulations, the maximum EPP absolute value difference is around 0.1 and average relative

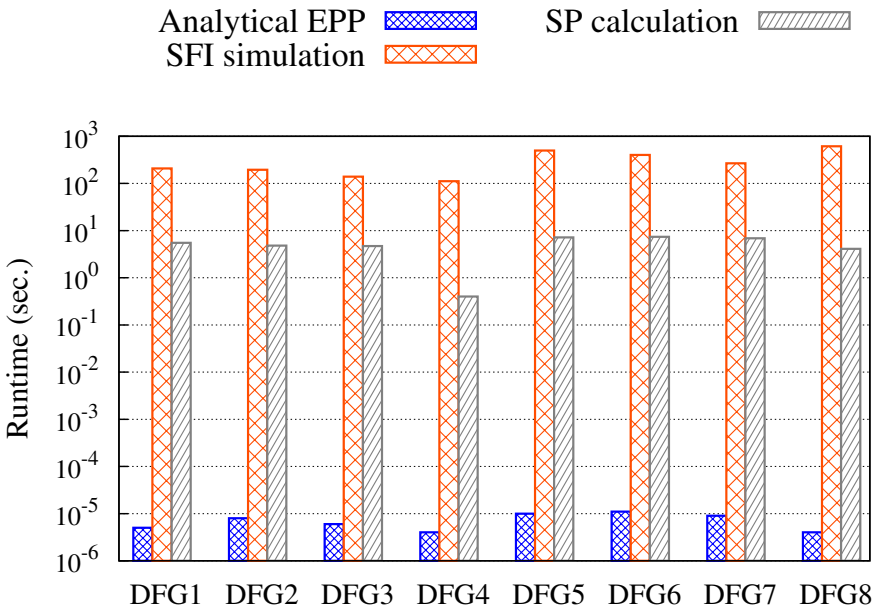


Figure 4.17: Runtime comparison of different DFG benchmarks

inaccuracy of our analytical approach is 8.64%. Considering that RTL design is in early design phase and much information of real implementation is not available, this inaccuracy is acceptable. It also shows that the previously mentioned convergent paths which result in dependency among input registers do not have significant impact on our approach.

The speedup results have two groups. In the first group, the runtime of our approach does not include signal probability runtime, which, however, is included in the second group. The average speedup ration for these two groups are 4.85E+07 and 80.72, respectively. Signal probability computation is a really time-consuming part, but fortunately, in real system design, SP values are available from power estimation. Reuse of these information makes our approach have high speedup compared with traditional fault simulation.

4.6.3 Case Study of OpenRISC IC FSM

To illustrate the vulnerability variance for different registers, we investigated the instruction cache FSM of the OpenRISC processor as a case study. This FSM has 40 primary inputs including one 32-bit address signal, and 8 control signals that interact with the cache RAM and bus interface to fetch cache-missed instructions. In addition, it has 42 internal registers

Table 4.6: PI signal probabilities with different workloads

Workloads	PI Signal Probabilities		
	icqmem_cycstb_i	tagcomp_miss	biudata_valid
BasicMath	0.89	**0.08**	0.054
Qsort	0.92	**0.13**	0.099
StringSearch	0.89	**0.01**	0.004
FFT	0.90	**0.02**	0.001

related to the FSM states, the instruction fetch address and other control signals for cache hit/miss evaluation.

For the investigation of register vulnerabilities with realistic workloads, we extracted the signal probabilities at PIs of the IC FSM from behavioral RTL simulation. Four typical workloads - *BasicMath, Qsort, StringSearch* and Fast Fourier Transform (*FFT*) from Mibench benchmarks [110] are selected to run on the processor. In Figure 4.18, we plot the vulnerabilities of several representative registers. The system failure probabilities within multiple time steps are computed with the formula in [148], which can combine the model checking results for each individual time step.

It is clearly shown that different registers in this FSM have non-uniform soft error vulnerabilities. In addition, with different workloads the register vulnerabilities manifest large variance, e.g. the register *cache_inhibit*. To further investigate the reasons for this invariance, in Table 4.6 we list the signal probabilities at some PIs of the IC FSM. Note that the PI signal *tagcomp_miss* is related to the cache miss rates and hence, the corresponding state transitions of the cache FSM. It is clear that the signal probabilities of this PI for workloads *BasicMath* and *Qsort* are larger (around 0.1) than that of *StringSearch* and *FFT*. This is also the reason why the register vulnerabilities in Figure 4.18 are close to each other within two different workload groups: (*BasicMath, Qsort*) and (*StringSearch, FFT*). In this case study, we can see that there is only 0.1 probability difference at the PI signal *tagcomp_miss*, but it leads to around 0.4 vulnerability difference for the internal register *cache_inhibit*.

Except for the different *quantitative* vulnerability values, we also investigated the *qualitative* trends across different running workloads. In Figure 4.19, we plot the vulnerabilities of the registers in the IC FSM, and sort them in a descending order according to their vulnerabilities respect to the *BasicMath* workload. Then the vulnerabilities of corresponding registers for the other three workloads are overlaid. From this figure we can see that there is large overlapped portion of the top-ranked registers in the four workloads. That is to say, although the *absolute* vulnerability values vary across different workloads, the *relative* vulnerability ranking of the registers are rather similar. This kind of qualitative information is very valuable for selective register protection in the RTL design with different running workloads.

4.6.4 Comparison with Related Work

To highlight our contributions, we also compare the features of the proposed method with several representative techniques on soft error evaluation [5, 6, 50, 78, 113, 115, 118, 149–151], both at circuit level and RTL. Among them only [50, 113, 149, 150] can provide probabilistic quantification of soft error vulnerabilities, and at the same time, take workload dependency into consideration. The techniques in [50, 149] handle only combinational circuits and are not applicable to the sequential analysis in control paths. The work in [113, 150] stay at circuit level, and lack the capability of RTL abstraction such as data type reduction to handle the designs with signals with large bitwidth.

As the RTL and circuit level techniques consider different types of benchmarks (i.e. behavioral FSMs *versus* gate-level netlists), we use the number of PIs and REGs to estimate the design size. Regarding the largest design considered in this work (*or1200_genpc* with 241 PIs and 95 REGs) and in related work (*s5378* with 35 PIs and 179 REGs in [151]), the runtime

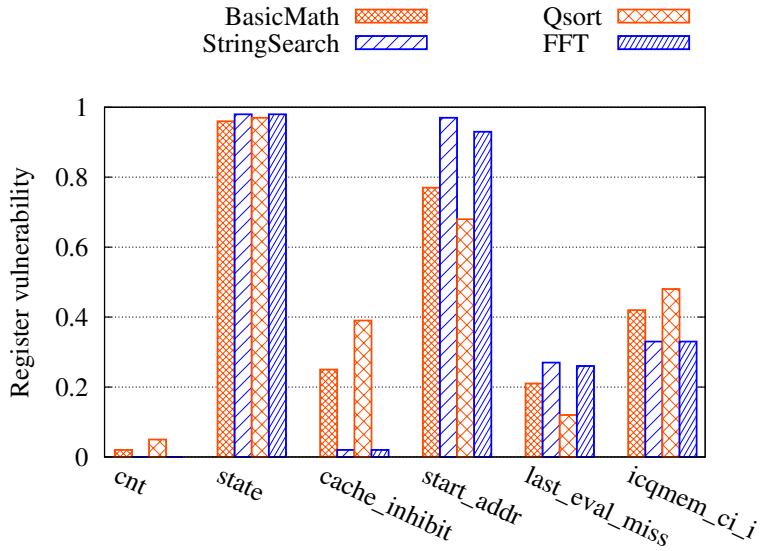


Figure 4.18: Vulnerabilities of representative registers in OR1200 IC FSM with four different workloads

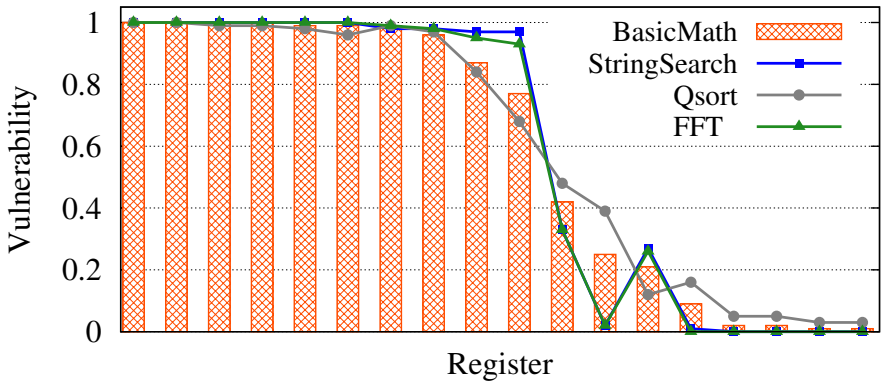


Figure 4.19: Vulnerability ranking of the registers sorted based on the *BasicMath* workload

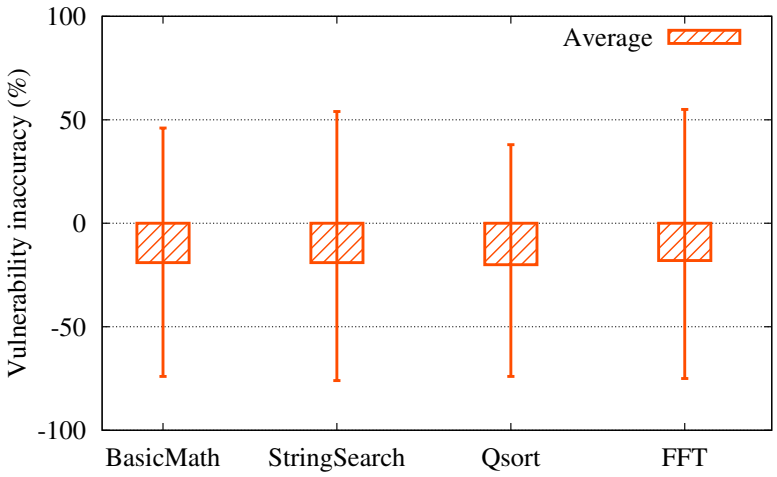


Figure 4.20: Inaccuracy of the register vulnerabilities in related work [5] due to random inputs assumption

of our proposed method is in a reasonable range to analyze large control modules in a typical embedded processor. Note that our method performs exhaustive exploration of the state space to avoid the approximate sequential circuit unrolling [113]. In the closest related work [5] on RTL soft error evaluation using formal technique, the authors can only use the random input assumption, because their error propagation probability is computed as $\frac{\#\text{ of SAT instances}}{2^{\#\text{ of inputs}}}$. To see how much inaccuracy can be introduced with this assumption, the signal probabilities of all inputs of the IC FSM benchmark are set to 0.5, then the corresponding register vulnerabilities are evaluated. We compare those obtained vulnerabilities with the values evaluated using signal probabilities extracted from real workloads in Section 4.6.3. Figure 4.20 shows that this simple assumption on average causes 20% inaccuracy of evaluated vulnerabilities, and the maximum deviation reaches around 80%. This significant inaccuracy arises due to the large difference with the input statistics as shown in Table 4.6. Therefore, it is essential to carefully take the workload dependency into consideration for accurate register vulnerability evaluation.

To summarize, the proposed method extends previous RTL error evaluation techniques by introducing probabilistic quantification and considering workload dependency.

4.7 Conclusion and Summary

For efficiently mitigating soft error in digital systems, it becomes increasingly important to evaluate and model soft error in early design phase to balance performance, cost and reliability. Most of previous SER modeling work are either at architectural level or circuit/logic level, which have either accuracy or complexity disadvantages. To face these challenges we propose new methodologies on RTL and behavior level soft error estimation, which can find a better balance between the complexity and accuracy of error analysis.

According to the different error propagation properties, the RTL control and data paths are analyzed with different methods. For the control path analysis, we employed a state transition system to represent the behavior of RTL design, and uses formal probabilistic model checking to handle the error propagation and masking effects. Efficient state space reduction techniques

are developed to improve the scalability of our approach to handle complex control modules in a typical embedded processor. For the data path, we have categorized basic operation groups and derived EPP rules for these operations, and then how to perform the overall EPP estimation was discussed. The experimental results based on DFG shows that compared with bit-level SFI method, our approach could achieve 7 orders of magnitude speedup, with less than 9% inaccuracy on average. For the controller, the experiments reveal that the registers in control paths have significantly non-uniform soft error vulnerabilities, which can be later exploited for cost-effective selective register protection.

In addition, we also discussed how to evaluate the vulnerabilities of variables and operations at behavioral level with similar model checking technique. The obtained vulnerability values will be used for selective protection in our error mitigation techniques in next chapters.

5 Reliability-aware Resource Allocation and Binding

As the error estimation techniques in previous chapters can provide accurate reliability evaluation of both the low level hardware modules and behavioral level constructs, in this chapter we focus on how to achieve efficient error mitigation by making use of such valuable vulnerability information. We propose a reliability-aware allocation and binding technique during the high level synthesis. This is based on the observation that for behavioral designs, especially control-flow intensive ones, variables and operations have non-uniform soft error vulnerabilities. The optimizations based on integer linear programming, as well as heuristic algorithm, are employed to incorporate the behavioral vulnerabilities into the register and functional unit binding phases to achieve cost-efficient error mitigation.

5.1 Introduction

During HLS, the automatic translation of a design from its behavioral description to a structural RTL implementation is performed. Generally the HLS process consists of three phases: *scheduling, allocation* and *binding*. Scheduling explores the concurrent operation execution to improve the performance of the RTL design, while the resource allocation and binding select necessary and efficient RTL resources to implement behavioral functionalities, and determine the mapping relation between the behavioral constructs to the allocated RTL resources [40]. Typically the allocation and binding process can be further divided into subtasks regarding Functional Units (FUs) and storage elements such as registers. The hierarchical feature and large flexibility of HLS can be fully leveraged to enhance the system reliability in a cost-efficient way [81, 82].

Due to the irregular structures of the sequential elements (registers in HLS) and the combinational logic cores (FUs in HLS), the low-cost coding techniques, which are widely adopted for memory protection, can not be applied here. As the full protection of such structures (e.g. triple modular redundancy) introduces very high overhead, selective hardening becomes the only viable solution for cost-efficient reliability enhancement [6]. To efficiently select the hardened registers and FUs in the RTL implementation, the vulnerabilities of the variables and operations in behavioral design, which will be mapped to these RTL resources during HLS, need to be carefully integrated.

The behavioral vulnerability values are related to the complexity of the control structures in the designs. Generally the behavioral designs can be divided into two categories: *Data-Flow Intensive* (DFI) designs and *Control-Flow Intensive* (CFI) designs [152]. The DFI designs are characterized by significant amount of arithmetic operations with few control dependencies, while the CFI ones contain many relational operations and have unbalanced branch execution

probabilities. Therefore, the variables and operations in CFI designs may manifest nonuniform vulnerabilities to soft error compared with that in DFI ones. This reveals the possibility to add vulnerability metric into the resource binding phase of CFI designs, so that more efficient soft error mitigation can be achieved. However, 1) how to identify the most vulnerable variables and operations to bind with reliable RTL resources, and 2) which registers and FUs should be protected to maximize the overall soft error mitigation, are very important questions that are not appropriately addressed in the literature so far.

In the recent reliability-aware HLS work [81–86], soft error-induced reliability is taken into account in addition to the traditional metrics such as performance, area and power. These works focus on data flow graph and only explore the potential of reliability improvement on the FU allocation and binding, but the important register binding process is ignored in these techniques. In addition, they leveraged only the RTL reliability information to explore the reliability enhancement, either by N modular redundancy [83–86], or multiple implementations of the RTL FUs with different area and reliability metrics [81, 82]. However, in addition to the RTL reliability metrics, the aforementioned behavioral vulnerability, i.e. the probability of system failure given that behavioral component being erroneous, is another essential factor which influences the efficiency of selective protection. Such important behavioral vulnerability information is totally ignored in the previous methods, which can lead to inappropriate priorities of register and FU protection and introduce unnecessary cost.

In this chapter we propose a *reliability-aware register and FU binding technique* based on Integer Linear Programming (ILP) to find the most cost-efficient selective protection scheme. In addition to the failure rates of the RTL resources, in our technique the behavioral vulnerabilities, previously presented in Section 4.5, are also taken into consideration. In particular, we make the following threefold contributions:

- *Reliability-aware register binding* that proposes an ILP-based optimization to bind vulnerable variables to hardened RTL registers;
- *Reliability-aware FU allocation and binding*, which extends the ILP formulation to bind operations to different implementations of FUs in the RTL library;
- *Heuristic binding* that exploits a hardening-efficiency guided greedy algorithm to significantly improve the scalability of the exact ILP method.

These features enable our framework to handle the general allocation and binding problems in HLS from the reliability perspective. Regarding runtime the heuristic reaches 76X speedup compared with the optimal ILP, while maintaining satisfactory accuracy.

To our best knowledge, this is the first attempt to explore the potential of register and FU binding taking soft error vulnerabilities of behavioral variables and operations into consideration. Compared with vulnerability-unaware binding, the experimental results show that our proposed technique can achieve much more efficient reliability enhancement, i.e. up to 85% higher reliability with the same area budget.

The organization of the rest of this chapter is as follows. Section 5.2 motivates the proposed work on vulnerability-based resource binding. Section 5.3 describes the ILP formulation of the reliability-aware register binding, and Section 5.4 focuses on the extension to FU allocation and binding. Section 5.5 discusses the heuristic binding algorithm and in Section 5.6 the experimental results are discussed. Finally Section 5.7 concludes the chapter.

5.2 Motivation

During HLS, behavioral variables are merged and bound to storage elements like registers, while operations are assigned to RTL FUs. The traditional allocation and binding problem focuses on the minimization of the number of registers or the aggregated FU cost, i.e. maximizing the resource sharing [39].

In the scope of reliability enhancement, cost-efficient resource allocation and binding can be performed by investigating two important factors. On the one hand, the behavioral variables and operations can have non-uniform error vulnerabilities, according to the error generation and propagation analysis described later. On the other hand, there exist significant binding flexibilities between the behavioral structures (i.e. variables and operations) and the RTL resources (i.e. registers and FUs) [40]. Therefore, by adjusting the binding relations, i.e. mapping most vulnerable behavioral structures to few RTL resources, and applying selective protection only to those resources, we can achieve cost-efficient reliability enhancement. In the following, we will take register binding as a running example to demonstrate this reliability-aware binding concept, which is similar for the case of FU binding.

To bind multiple variables into a shared register, the information on variable lifetime is necessary. The *lifetime* of a variable is defined as the set of states in which the variable is alive, including the state in which this variable is defined, every state in which it is used as an operand of another operation, and all the states on the path between the definition state and the usage state [39].

In the traditional register binding phase, the register sharing among different variables is maximized to reduce the number of storage components. This optimization requires one basic constraint to be satisfied: one register can be shared only by the *compatible* variables. Considering a scheduled control and data flow graph, two variables are defined as compatible when i) they do not have overlapping lifetimes, or ii) their lifetimes extend over mutually exclusive control paths. In this work, the register binding is performed in each *behavioral function*, which is always represented by high level programming language, e.g. function in C, and both the variable lifetime and compatibility information are extracted using the LegUp HLS framework [153].

5.2.1 Non-uniform Soft Error Vulnerabilities

The *vulnerability* of a variable is defined as the probability that the impaired content of this variable by soft errors will cause an erroneous value at the function output. The lifetime of a variable is an important factor influencing its vulnerability, as soft error is a random process over the entire period of the function execution. Intuitively, the variable with long lifetime is probable to have high soft error vulnerability. However, it is not necessarily proportional to its lifetime, as there are other factors such as the branch probabilities and functional dependencies which also impact the vulnerability value [120]. One simple example is a variable defined and used near the function exit, even if it has relatively short lifetime, its soft error vulnerability can be rather large due to the short error propagation paths.

To better illustrate this phenomenon, Figure 5.1 shows the deviation of variable lifetimes from the vulnerabilities for the *barcode* benchmark [154]. Here the lifetimes are normalized to the largest value to be comparable with vulnerability values, which are from our comprehensive

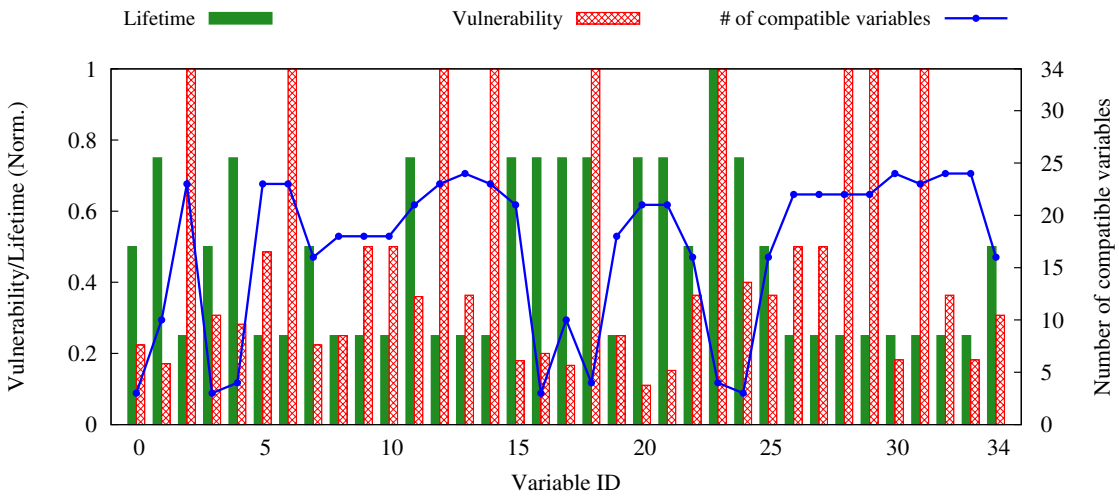


Figure 5.1: The lifetime, vulnerabilities and compatibilities of the variables in the *barcode* benchmark

analysis previously detailed in Section 4.5. This clearly shows that the vulnerability values of these variables manifest significant variance, and more importantly, different trends with their lifetimes. Therefore, the lifetime of a variable is not an appropriate metric to use as its soft error vulnerability, and more sophisticated analysis, considering error propagation and masking, is indispensable.

5.2.2 Non-unique Binding Solutions

Given the compatibility information among variables, minimization on the number of necessary registers can be performed. However, even for the minimum number of registers, there are still not unique binding solutions between variables and registers [39]. This is due to the fact that these variables manifest variant *mobility*, i.e. the variable compatible with more variables has higher mobility, and therefore can be allocated to different registers while the total number of registers remains the same. Figure 5.1 also shows the number of compatible variables for each variable in the *barcode* benchmark, which can be seen as a measurement of their mobilities. We can see that many highly vulnerable variables also have high mobilities. This means that there is a significant potential to improve the register binding quality with respect to reliability, i.e. binding more vulnerable variables to fewer registers, therefore using less register protection to achieve more efficient soft error mitigation.

5.3 Reliability-aware Register Binding

Given the variable and operation vulnerabilities obtained in Section 4.5, we can integrate both the behavioral and RTL reliability metrics into the allocation and binding phases. In the following, the formulation of reliability-aware register binding is first introduced and then extended to the case of FU binding in the next section.

The aim of reliability-aware register binding is to use the minimum subset of protected registers to cover the maximum variable vulnerabilities, i.e. “jamming” as many vulnera-

ble variables as possible to the protected registers. This will result in maximum soft error mitigation at the minimum costs. To achieve this, we use an optimization based on integer linear programming due to its advantage of obtaining optimal solutions and easy extension to incorporate additional constraints and objectives. In the following the ILP formulation is explained.

5.3.1 Definitions and Notations

Let $v_1, v_2, \ldots, v_N$ represent the total N variables in behavioral description, and $r_1, r_2, \ldots, r_R$ the total R available registers. In addition, the following notations are employed to facilitate the ILP formulation:

- $\mathbf{Vul^v}$: a vector of size N, and Vul_i^v is a constant with real value in the range $[0, 1]$, which is the vulnerability of variable v_i obtained in Section 4.5 scaled by $active\_states_i^v/total\_states$ to take the variable lifetime into consideration, i.e. the individual vulnerability contribution of multiple variables bound to a single register;
- $\mathbf{P}$: a vector of size R, and P_j is a binary value 1/0, representing whether register r_j should be protected or not;
- $\mathbf{Vul^r}$: a vector of size R, and Vul_j^r is a real value, representing the sum of the vulnerabilities of all variables bound to register r_j;
- $\mathbf{M^{REG}}$: an $N \times R$ matrix and $M_{i,j}^{REG}$ is a binary value 1/0, representing whether variable v_i is bound to register r_j or not.

$$\mathbf{Vul^v} = [Vul_1^v \ldots Vul_N^v],\ \mathbf{M^{REG}} = \begin{bmatrix} M_{1,1}^{REG} & \cdots & M_{1,R}^{REG} \\ \vdots & \ddots & \vdots \\ M_{N,1}^{REG} & \cdots & M_{N,R}^{REG} \end{bmatrix},\ \mathbf{P} = \begin{bmatrix} P_1 \\ \vdots \\ P_R \end{bmatrix}$$
$$\mathbf{Vul^r} = \mathbf{Vul^v} \cdot \mathbf{M^{REG}} = [Vul_1^r \ldots Vul_R^r] \tag{5.1}$$

For the register binding problem, there are two basic constraints which should be satisfied as prerequisites:

- *Uniqueness*: each variable v_i should be bound to one and only one register

$$\sum_{j=1}^{R} M_{i,j}^{REG} = 1 \text{ for each variable } v_i \tag{5.2}$$

- *Compatibility*: two incompatible variables v_{i1} and v_{i2} (i.e. with overlapped lifetime) cannot be bound to the same register

$$M_{i1,j}^{REG} + M_{i2,j}^{REG} \leq 1 \text{ for each register } r_j \tag{5.3}$$

There are also two additional metrics used in our ILP formulation:

- *Reliability*: the overall variable vulnerabilities covered by the protected registers is

$$\mathbf{Vul^v} \cdot \mathbf{M^{REG}} \cdot \mathbf{P} = \mathbf{Vul^r} \cdot \mathbf{P} = \sum_{i=1}^{N} \sum_{j=1}^{R} Vul_i^v \cdot M_{i,j}^{REG} \cdot P_j \tag{5.4}$$

Maximizing this coverage corresponds to minimizing the system failure rate SFR^{REG} contributed by the registers in the RTL implementation, i.e. maximizing the register reliability. Given the failure rate of a single unprotected and protected register $\lambda_{REG}, \lambda'_{REG}$ (constant values from the RTL component library), the overall register failure rates can be expressed as

$$SFR^{REG} = \lambda_{REG} \cdot \sum_{i=1}^{N} Vul_i^v - (\lambda_{REG} - \lambda'_{REG}) \cdot \mathbf{Vul^r} \cdot \mathbf{P} \tag{5.5}$$

- *Cost*: the number of protected registers can be formulated as

$$\sum_{j=1}^{R} P_j \tag{5.6}$$

Given the cost of unprotected register c_{REG} and protected register c'_{REG}, minimizing this number is equivalent to minimize the overall cost of the registers:

$$Cost^{REG} = c_{REG} \cdot R + (c'_{REG} - c_{REG}) \cdot \sum_{j=1}^{R} P_j \tag{5.7}$$

5.3.2 Register Binding Optimization

For the register binding, on the one hand, associating one register with each variable suffices, which gives the upper bound on the number of registers. On the other hand, the *Uniqueness* and *Compatibility* constraints limit the lower bound. To explore the potential of reliability-aware binding, we perform a two-pass optimization.

Minimum Register Optimization

In this optimization, the number of register R should be minimized, which means the R value is not fixed and the dimension of binding matrix $\mathbf{M^{REG}}$ is variant. To overcome this difficulty, we assume at the beginning the total number of register is set to the upper bound, i.e. $R = N$. Then with the constraints in Equation (5.2)(5.3), we try to maximize the number of all-0 columns in $\mathbf{M^{REG}}$, which is equivalent to minimize the number of registers:

$$\max \sum_{j=1}^{R} NOR_{i=1}^{N} M_{i,j}^{REG} \tag{5.8}$$

where $NOR_{i=1}^{N}$ represents the NOR function of N Boolean variables. As ILP requires all constraints and objective to be linear, we use the similar method as in [155] to convert the Boolean function NOR into linear constraints:

$$f = NOR(i_1, \ldots, i_n) \Rightarrow \begin{cases} f \leq 1 - (i_1 + \cdots + i_n)/n \\ f \geq 1 - (i_1 + \cdots + i_n) \end{cases} \tag{5.9}$$

Reliability-aware Optimization

Given the number of available registers R (e.g. the minimum value), we can perform the following optimization with the basic constraints in Equations (5.2)(5.3):

- *Cost-constrained*: maximize the overall coverage of variable vulnerabilities with the number of protected registers as the constraint R_c:

$$\begin{aligned} &\max \sum_{i=1}^{N}\sum_{j=1}^{R} Vul_i^v \cdot M_{i,j}^{REG} \cdot P_j \\ &\text{subject to } \sum_{j=1}^{R} P_j \leq R_c \end{aligned} \tag{5.10}$$

- *Reliability-constrained*: minimize the number of protected registers with the overall coverage of variable vulnerabilities as the constraint Vul_c:

$$\begin{aligned} &\min \sum_{j=1}^{R} P_j \\ &\text{subject to } \sum_{i=1}^{N}\sum_{j=1}^{R} Vul_i^v \cdot M_{i,j}^{REG} \cdot P_j \geq Vul_c \end{aligned} \tag{5.11}$$

Complexity Analysis

The ILP complexity can be represented with the numbers of Linear Programming (LP) variables and constraints (assume N behavioral variables and R registers):

- Minimum Register Optimization ($N = R$): from Equations (5.2), (5.3), (5.8), (5.9) the number of LP variables is $O(N^2)$ and that of constrains is $O(N^3)$;
- Reliability-aware Optimization ($N >= R$): from Equations (5.2), (5.3), (5.10)/(5.11) the number of LP variables is $O(R \cdot N)$ and that of constrains is $O(R \cdot N^2)$.

Therefore, for our two-pass optimization the number of LP variables is $O(N^2)$, and that of constrains is $O(N^3)$.

5.4 Functional Unit Allocation and Binding

Compared with the register binding presented in the previous section, the reliability-aware FU allocation and binding is a more sophisticated problem due to the following reasons:

- There are *multiple* types of FUs to execute different kinds of behavioral operations, rather than a *single* type of register to store behavioral variables;
- In addition to the timing compatibilities from scheduling results, there are also type compatibilities of the operations, which determines whether the behavioral operations can be executed by specific types of RTL FU components. For instance, a comparison operation can be bound to either an RTL comparator or an ALU, but not an adder.

To generalize the ILP formulation of reliability-aware register binding to multiple types of FU allocation and binding, we need to account for these difference. First there are several important terms which need to be clearly defined:

- *Operation*: the instance of the behavioral operations in the design, e.g. $op_1, op_2, \ldots$;
- *Operation type*: the type of the behavioral operations, e.g. ADD (addition), CMP (comparison) and MUL (multiplication). Each operation has only one operation type;

- *FU type*: the type of the RTL functional units, e.g. adder (+), comparator (>, < ...) and ALU;
- *FU instance*: in an RTL design there can be several instances of the same FU type, e.g. I_1^+, I_2^+, I_1^{ALU}, ...;
- *Resource*: a specific RTL implementation of an FU instance, which has different area, power and reliability metrics, e.g. an original or hardened implementation of the FU instance I_1^+;

Assume there are overall N^{OP} operations in a behavioral function, and total R^{FU} resources from an RTL library to execute these operations. Similar to the register binding in Section 5.3, the following notations of FU binding are used, which can be generally grouped into constant and variable categories.

Constant matrices:

- $\mathbf{Vul^{op}}$: a vector of size $1 \times N^{OP}$, and the operation vulnerability Vul_i^{op} is a real value in the range $[0, 1]$. Similar to the variable vulnerabilities, the operation vulnerability used here is also scaled by $active\_states_i^{op}/total\_states$ to consider the operation execution time;
- $\lambda^{\mathbf{r}}$: a vector of size $R^{FU} \times 1$, and λ_j^r is a real value, representing the failure rate of the resource r_j in the RTL library;
- $\mathbf{C^r}$: a vector of size $R^{FU} \times 1$, and C_j^r is a real value, representing the cost (e.g. area) of resource r_j in the RTL library;

Variable matrices:

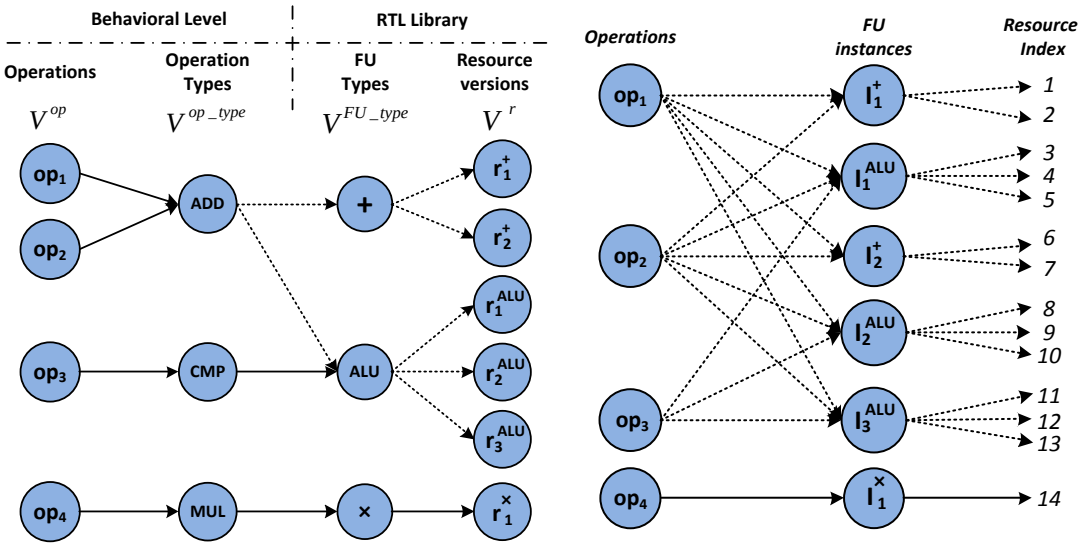
- $\mathbf{M^{FU}}$: an $N^{OP} \times R^{FU}$ matrix and $M_{i,j}^{FU}$ is a binary variable, representing whether operation op_i is bound (mapped) to resource r_j or not.
- $\mathbf{B^r}$: a vector of size $1 \times R^{FU}$ and B_j^r is a binary variable, representing whether resource r_j is bound or not in the synthesized RTL. Because several behavioral operations may be bound to the same RTL resource, the element of $\mathbf{B^r}$ is obtained by the *OR* function of all the elements in column j of matrix $\mathbf{M^{FU}}$, i.e. $B_j^r = OR_{i=1}^{N^{OP}} M_{i,j}^{FU}$.

$$\mathbf{M^{FU}} = \begin{bmatrix} M_{1,1}^{FU} & \cdots & M_{1,R^{FU}}^{FU} \\ \vdots & \ddots & \vdots \\ M_{N^{OP},1}^{FU} & \cdots & M_{N^{OP},R^{FU}}^{FU} \end{bmatrix}, \mathbf{B^r} = \begin{bmatrix} B_1^r \cdots B_{R^{FU}}^r \end{bmatrix} \tag{5.12}$$

5.4.1 FU Allocation and Binding Space Determination

Given a behavioral design and an RTL library with different versions of FU implementation, we need to determine the search space of the reliability optimization problem, i.e. the number of allocated RTL resources to contain all the possible binding options. Specifically, the size R^{FU} of the binding matrix $\mathbf{M^{FU}}$ should be derived and proper binary ILP variables should be assigned to its elements as well.

We propose a three step procedure to determine the binding search space while considering the operation/FU type compatibilities. For the running example in Figure 5.2, at the behavioral level there are four operations with three types: addition (ADD), comparison (CMP) and multiplication (MUL). On the RTL side there are three FU types: adder (+), ALU and



(a) Example of mapping and binding graph

(b) Binding space constructed from MBG

Figure 5.2: An example of resource allocation and binding space determination

multiplier (×), and for each FU instance there are several kinds of resources implementing the same functionality, but with different characteristics such as area, power and reliability. The following procedure determines how many FU instances as well as FU resources are necessary to cover the entire binding space.

Mapping and Binding Graph (MBG) Construction

First we construct a four-level mapping and binding graph $G = (V, E)$, which is directed acyclic, as shown in Figure 5.2(a). The vertices grouped in four levels, namely V^{op}, V^{op_type}, V^{FU_type} and V^{r}, represent the operations, operation types at behavioral level and FU types, resource versions in the RTL library. The edges E represent the mapping and binding relations between vertices in neighbouring levels. The solid edges represent *fixed* mapping relation between its head and tail vertices, and the dotted edges represent *optional* binding relations.

Backward MBG Traversal

Then the backward traversal of MBG is performed to determine how many FU instances are necessary to cover all the possible binding options. For each vertex in V^{FU_type}, we extract all its *predecessors*, including the direct and indirect ones [40]. The number of V^{op} vertices in the predecessors determines the number of such FU instances that should be included in the binding search space. For instance, the predecessors of the vertex v_{ALU} include three V^{op} vertices: v_{op_1}, v_{op_2} and v_{op_3}. Hence, there should be three ALU instances to cover *all* the possible binding options. Similar analysis shows that two adder instances and one multiplier instance are needed, as shown in Figure 5.2(b). As there are 2 (3) different versions of RTL implementations for each adder (ALU) instance, in total we have 14 resources as the binding candidates.

Binding Space Determination

Finally each element value in the binding matrix is determined. Initially all the elements of $\mathbf{M^{FU}}$ are assigned to 0. Then for each RTL resource r_j, if it can execute operation op_i and there are multiple binding candidates (dotted edges) for op_i, a binary ILP variable is assigned to the element $M^{FU}_{i,j}$. If there is only one single candidate (i.e. r_j itself), constant value '1' is assigned, e.g. $M^{FU}_{4,14} = 1$ because operation op_4 will be definitely bound to resource r_{14}. Note that the limited binding options among behavioral operations, FU instances and specific RTL resources, as shown in Figure 5.2(b), make the matrix $\mathbf{M^{FU}}$ to be sparse, and the number of binding ILP variables is much smaller than the product of N^{OP} and R^{FU}.

5.4.2 Constraints and Objective

Similar to the register binding formulation, there are also two basic constraints to be satisfied for the FU binding problem:

- *Uniqueness*: each operation op_i should be bound to one and only one resource.

$$\sum_{j=1}^{R^{FU}} M^{FU}_{i,j} = 1 \text{ for each operation } op_i \tag{5.13}$$

- *Compatibility*: As the functional type compatibilities are already considered during the construction step of binding matrix $\mathbf{M^{FU}}$ in Section 5.4.1, only scheduling compatibilities need to be formulated here. Two operations op_{i1} and op_{i2} are *incompatible* when they are concurrently executed in the same time step. Two incompatible operations should not be bound to the same resource which can execute these two operations.

$$M^{FU}_{i1,j} + M^{FU}_{i2,j} \leq 1 \text{ for each resource } r_j \tag{5.14}$$

Both the reliability and cost metrics are considered in our ILP formulation:

- *System Failure Rate (SFR^{FU})*: the system failure rate contributed by the FUs in the synthesized RTL, which is the sum of the products of RTL resource failure rate λ^r_j (error generation) and the behavioral vulnerabilities Vul^{op}_i (error propagation) of the bound operations ($M^{FU}_{i,j} = 1$):

$$SFR^{FU} = \mathbf{Vul^{op}} \cdot \mathbf{M^{FU}} \cdot \lambda^{\mathbf{r}} = \sum_{j=1}^{R^{FU}} \sum_{i=1}^{N^{OP}} Vul^{op}_i \cdot M^{FU}_{i,j} \cdot \lambda^r_j \tag{5.15}$$

- *Cost*: the aggregated cost of bound RTL resources. Due to the FU sharing, a multiplexer (MUX) has to be inserted before an input port of an FU whenever multiple registers feed data to this port. In our reliability-aware binding problem, the FU cost, and the MUX cost at the input port of FUs can be formulated as follows:

$$Cost^{FU} = \mathbf{B^r} \cdot \mathbf{C^r} = \sum_{j=1}^{R^{FU}} B^r_j \cdot C^r_j \tag{5.16}$$

$$Cost^{MUX} = D^{MUX} \sum_{j=1}^{R^{FU}} \sum_{p=1}^{Port_j} \left(\sum_{reg=1}^{REG} z_{reg,j,p} - B^r_j \right) \tag{5.17}$$

D^{MUX} is the cost of a MUX unit. Assume resource r_j has $Port_j$ number of input ports, and var is the p^{th} operand of operation op_i. There must be one connection (i.e. $z_{reg,j,p} = 1$) from register reg, holding the value of variable var, to the p^{th} port of FU r_j which executes the operation op_i. The employed MUX formulation is similar to [156], and considers both the register and FU binding simultaneously.

5.4.3 Reliability-aware FU Binding Optimization

Quantification of the Design Space Boundaries

Given the operation vulnerabilities as well as the cost and failure rate metrics of the resource in RTL library, we can separately determine the lower bound of SFR^{FU} or cost. This can give the designer good estimation of the characteristics of the synthesized RTL. Moreover, it is a starting point of further design space exploration on the trade-off between cost and SFR. Two kinds of optimization can be performed with the basic constraints in Equations (5.13)(5.14) and objectives in Equations (5.15), or (5.16)(5.17):

- *Minimization of* SFR^{FU}: obtain the minimum failure rate SFR^{FU}_{min} of the RTL implementation without cost constraint;
- *Minimization of cost*: obtain the minimum cost $Cost_{min}$ of the RTL implementation without any constraint on SFR^{FU}.

Reliability-aware Optimization

After the lower bounds of cost and SFR^{FU} are obtained, two kinds of optimization can be performed with the basic constraints in Equations (5.13)(5.14):

- *Cost-constrained*: minimize the SFR^{FU} of the RTL implementation with the cost constraint $C_c = (1 + \alpha) \cdot Cost_{min}$ (α is a user-specified coefficient of cost overhead);
- *SFR-constrained*: minimize the overall cost of the RTL implementation with the SFR constraint $SFR^{FU}_c = (1 + \beta) \cdot SFR^{FU}_{min}$ (β is a user-specified coefficient of allowable SFR^{FU} increase).

5.5 Vulnerability Compaction and Heuristic Resource Binding

For both register and FU binding, either the cost-constrained or the reliability-constrained optimizations can be performed according to the user requirements. However, it has the disadvantage that whenever the constraint changes, the entire optimization has to be rerun. In addition, ILP requires integer solutions and generally is a computation-expensive problem. The ILP-based resource binding is an NP-complete problem, especially for the resource constrained scenarios [157], and hence is mainly applied to the designs with moderate size [40]. To overcome these issues, we propose to perform *vulnerability compaction* to avoid multiple-run optimization with different cost/reliability constrains, and use an efficient *hardening heuristic* to explore the trade-off between reliability and cost, as shown with details in Algorithm 5.1.

The motivation for the optimization on vulnerability compaction is to find the optimal register and FU sharing with regard to vulnerability concentration. The basic idea is to sort

the available RTL components (i.e. registers and FUs) in a descending order according to their accumulated vulnerabilities, i.e. concentrate the component vulnerability distribution as much as possible. Therefore, we need to maximize the vulnerabilities covered by the first one component, the first two components, and until first R components, which can be formulated as following:

$$\begin{aligned} & Vul_j^{RTL} = \sum_{i=1}^{N^{beh}} Vul_i^{beh} \cdot M_{i,j}^{RTL} \\ & \max\ Vul_1^{RTL} + (Vul_1^{RTL} + Vul_2^{RTL}) + \cdots + (Vul_1^{RTL} + \cdots + Vul_{I^{FU}}^{RTL}) \\ \Rightarrow & \max \sum_{j=1}^{I^{FU}} \sum_{k=1}^{j} Vul_k^{RTL} \end{aligned} \tag{5.18}$$

in which the N^{beh}, Vul_i^{beh}, and $M_{i,j}^{RTL}$ correspond to either behavioral variables or operations as in Equation (5.4) or (5.15). The benefits of the compaction are twofold: i) this optimization is independent of the RTL component characteristics and need to be run only once for different cost/reliability constraints; ii) a single run of the compaction optimization can find the solution much faster than a single run of cost/reliability-constrained optimization, because we remove the tight cost (reliability) constraints and just change the weight of vulnerabilities of different RTL components in the objective function. After solving this optimization problem, the list of RTL components are ordered in a descending order based on their concentrated vulnerabilities.

In the subsequent RTL hardening step, to decide which registers should be protected within the cost constraints, we just need to select the top registers in previous descending list. For FU hardening, the selection of appropriate implementation in the RTL library for each FU instance is performed based on its concentrated vulnerabilities. The main goal of the RTL binding is to maximize the FU hardening efficiency, i.e. achieving large reduction of failure rates with small hardening overhead. For this purpose, a greedy algorithm is proposed and described in details in Algorithm 5.1. First we evaluate the efficiency for each possible hardening case with the finest cost granularity in the RTL library, i.e. from one FU version to its nearest hardened version (Line 6 to 15). Then in Line 16 to 25 the efficiency calculation considers both the reliability of RTL FUs (error occurring rates) and the behavioral vulnerability (error propagation probability).

Assume there are total N behavioral operations, and in the worst case without sharing, N FU instances are necessary. If the maximum number of versions for RTL FUs is V, the time complexity of this greedy heuristic is $O(N \cdot Vlog(N \cdot V))$. This is because the part of sorting $N \cdot V$ hardening cases is $O(N \cdot Vlog(N \cdot V))$, and the part of selecting hardening candidates is $O(N \cdot V)$.

5.6 Experimental Results

The proposed approach is implemented and applied to the CDFGs extracted from a set of HLS benchmarks on a workstation with Intel Xeon E5540 2.53 GHz and 16 GB RAM. The square-root approximation function *sra_func* [39], the barcode reader *barcode* [154], the send process of the X.25 communications protocol *send_X25*, QRS state machine *qrs*, the ADPCM coder and decoder, and the quantization and coding function *gsm_qc* [158] are CFI benchmarks and

Algorithm 5.1 Vulnerability Compaction and Heuristic Binding for Efficient Reliability Enhancement

Input: CDFG of the behavioral function, vulnerabilities of each variable/operation
Input: scheduled state information of each behavioral variable/operation, and different FU implementations in an RTL library
Input: cost budgets for register and FU hardening
Output: the sequence of hardening specific registers and FUs to achieve maximum efficiency of reliability enhancement

Perform the vulnerability compaction optimization on registers and FU instances, and return the sorted list of registers $list\_REGs$ and FU instances $list\_FU\_inst$.

$RTL\_lib\_hardening\_info = \emptyset$
for each FU type t^{FU} in the RTL library **do**
    Sort different versions of t^{FU} in an ascending order regarding the cost;
    for each version v in this sorted list **do**
        $\Delta Cost = Cost_{v+1} - Cost_v$
        $\Delta FIT = FIT_v - FIT_{v+1}$ ▷ Hardening introduces additional cost with reduced failure rate
        Harding efficiency $\eta^{FU}(t^{FU}, v, v+1) = \frac{\Delta FIT}{\Delta Cost}$
        Add $\eta^{FU}(t^{FU}, v, v+1)$ to $RTL\_lib\_hardening\_info$
    end for
end for

$FU\_inst\_hardening\_info = \emptyset$
for each instance j in $list\_FU\_inst$ **do**
    Its FU type is t_j and the concentrated vulnerability is Vul_j^{inst}
    for each $\eta^{FU}(t^{FU}, v, v+1)$ in $RTL\_lib\_hardening\_info$ **do**
        if $t_j == t^{FU}$ **then**
            $\eta_j^{inst}(t_j, v, v+1) = Vul_j^{inst} \times \eta^{FU}(t^{FU}, v, v+1)$
            Add $\eta_j^{inst}(t_j, v, v+1)$ to $FU\_inst\_hardening\_info$
        end if
    end for
end for

Sort $FU\_inst\_hardening\_info$ in an descending order of η_j^{inst}
Within the given cost budgets of register and FU hardening, select the top items of the $list\_REGs$ and the sorted $FU\_inst\_hardening\_info$ as hardening candidates.

the differential equation solver *diffeq* [154] is one DFI benchmark for comparison. The open-source HLS tool LegUp [153] and the LP solver CPLEX are employed for scheduling and ILP optimization, respectively. The RTL area and power values are reported after logic synthesis by Design Compiler based on the 45 nm Nangate library.

5.6.1 Work Flow

The overall reliability-aware register and FU binding flow is illustrated in Figure 5.3. First, the behavioral description of the design is processed by the front-end of the LLVM compiler infrastructure [159] and the platform-independent Intermediate Representation (IR) are obtained. Second, our script parses the IR and generates the vulnerability evaluation model. Then the model checker PRISM analyzes the model and extracts the variable and operation vulnerabilities. At the same time, the LegUp HLS tool performs the scheduling process and the variable lifetime and operation execution states are obtained accordingly. Then we carry out the reliability-aware optimization for both register and FU binding. In case that the number of register is not specified, ILP is used to obtain the minimum number; otherwise the variable vulnerabilities are taken into consideration and the optimization on reliability-aware register binding is carried out to obtain the binding and protection solutions. Similarly, the lower bounds of SFR and cost are obtained first as described in Section 5.4.3, then the cost-reliability trade-off is investigated by selective hardening the vulnerable FUs. Finally the combined analysis of register protection and FU hardening is carried out to evaluate their individual contributions to the overall system SER.

5.6.2 Characterization of the RTL Resources

For our experimental evaluation, an RTL component library with soft error characterization is used. In this library, different implementations of the same resource type with different area, power and SER values are available. The SER metric is expressed with FIT rate (Failure In Time), i.e. 1 FIT is equal to 1 error per billion operational hours.

For the register hardening, we employed the DICE technique [61], and used the data from real radiation testing [33]. With 80% area and 60% power overheads, the SER of the DICE register can be reduced 10X compared with the unprotected one. For the FU hardening, we investigated variety of techniques including gate sizing, voltage scaling, and transient filter insertion. Among them, transient filters provide a considerable SER reduction by imposing small area and power overheads [4]. Using this method, we have implemented three different versions of each RTL FU type with various masking factors. Then, the overall area, power, and SER of each version were evaluated.

For SER computation of each version of RTL FUs, we employed a hierarchical approach for error generation and propagation analysis based on [53]. First, the effect of soft errors at device level is analyzed with combination of 3D-TCAD simulation and current injection on transistors at SPICE netlist using a commercial FIT rate analysis tool [160] with 45 nm Nangate library. As shown in Figure 5.4, for each combinational cell this tool provides the distribution of pulses of various widths, depending on the output load seen by the cell. Then, the current pulses at the cell output are propagated from the error site to the primary outputs or downstream flip-flops. During the error propagation, three important masking effects, namely electrical, logic

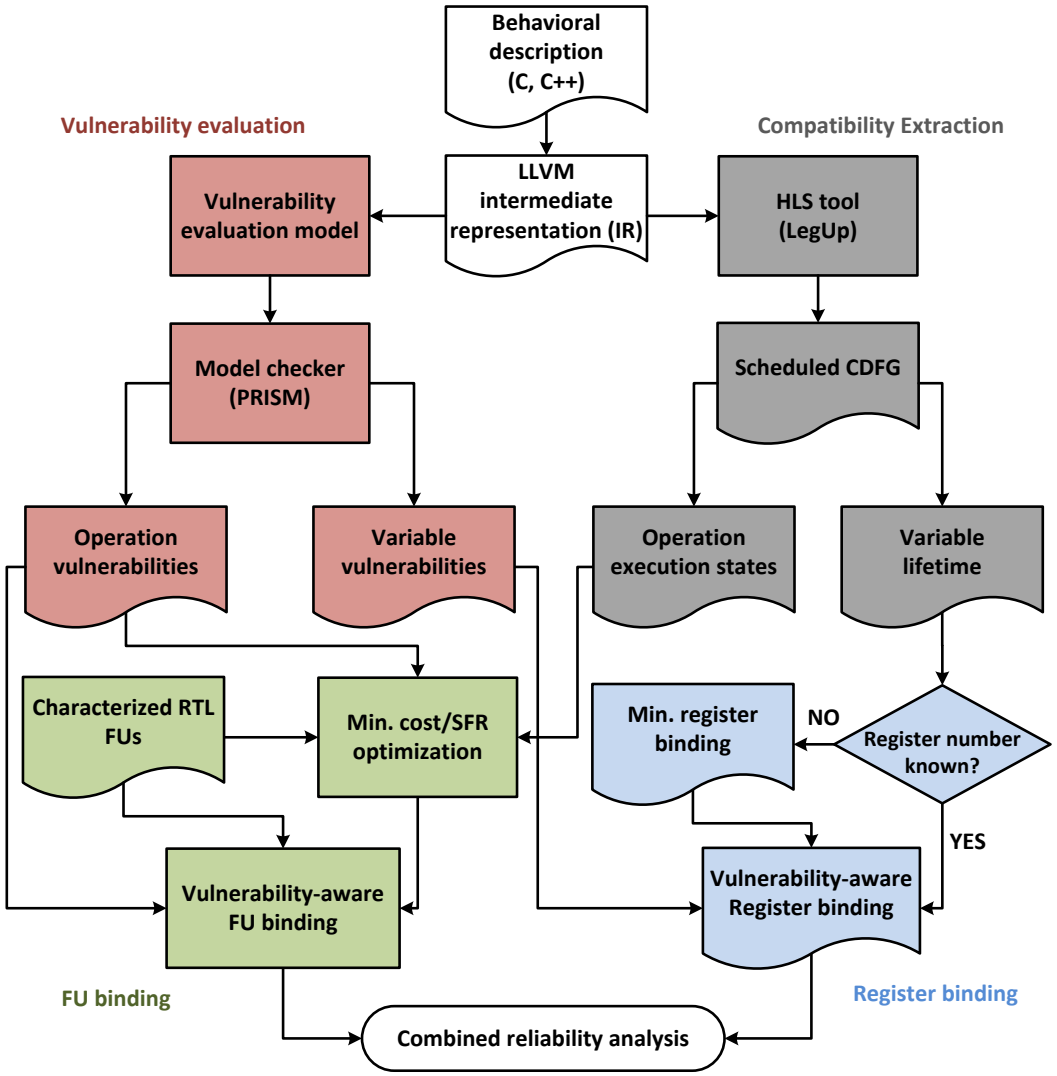
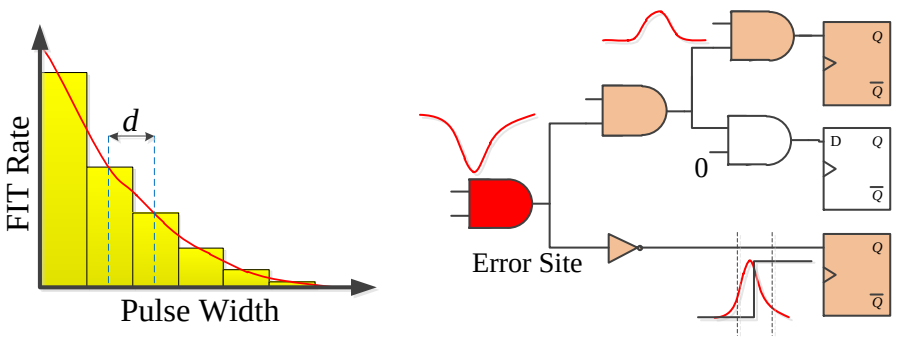


Figure 5.3: Reliability-aware register and FU binding flow



(a) FIT Rate of each pulse width interval for the error site gate

(b) Error propagation with electrical, logic and timing window masking

Figure 5.4: SER characterization of the RTL FU circuits

Table 5.1: Area, power and reliability metrics of different versions of resource in the employed RTL library

RTL metrics	Flip-Flop		32bit-Adder			32-bit-XOR			32-bit-Comparator		
	regular	DICE	v1	v2	v3	v1	v2	v3	v1	v2	v3
Area (μm^2)	4.52	8.14	421.79	425.16	428.53	121.41	124.67	127.94	109.66	109.76	109.87
Power (μW)	0.42	0.68	324.30	327.74	331.19	62.48	65.82	69.16	69.11	69.21	69.32
Failure Rate ($\times 10^{-3}$ FIT)	0.16	0.016	14.87	10.07	6.03	7.12	4.85	3.25	0.48	0.32	0.18

and timing windows masking are all taken into account. The obtained SERs are expressed as the number of FIT. In Table 5.1 the area, power and reliability values of several representative resources in the employed RTL library are listed. For entire system, the FIT rate can be calculated by summing the FIT rates of each individual components in the system [3], and we compute the system failure rates as $SFR = SFR^{REG} + SFR^{FU}$ using Equations (5.5)(5.15).

As soft error FIT rate and Mean Time to Failure (MTTF) of the system are inversely related [3], in this work we use the MTTF of the synthesized RTL system as the *reliability metric* to evaluate different allocation and binding solutions. For each benchmark, the binding solution with SFR_{min} corresponds to the maximum achievable reliability, i.e. MTTF_{max}, and to illustrate the obtained results better, the obtained MTTF after each evaluation is normalized to this MTTF_{max}. The *cost* used in the experiments is the area of the bound RTL resources, and starting from the binding scenario with minimum area value, we evaluate the normalized reliability that can be achieved with additional area budget.

5.6.3 Register Binding Results Analysis

To show the effectiveness of the proposed reliability-aware binding technique, we first evaluated the register binding and FU binding separately, and then the combined binding. Two selective protection scenarios are investigated for the register binding:

- *Vulnerability-unaware baseline*: as there is no previous work investigating this particular problem, we construct "the best effort" approach for selective protection. In this scenario the actual vulnerabilities of the variables are unknown, and the registers are sorted in a descending order according to the number of variables bound to each register, then based on the protection budget, the top registers in this list are hardened;
- *Vulnerability-aware binding*: this scenario corresponds to the optimization step for register vulnerability compaction, and the registers are sorted according to the solution of ILP optimization.

We then calculate the register reliability value by selectively protecting registers in each scenario, and normalize it to the maximum achievable reliability, i.e. when all registers are hardened in each benchmark.

The results in Figure 5.5 show that for all the CFI benchmarks, with the same hardware overhead the vulnerability-aware binding outperforms the baseline. However, for the DFI benchmark *diffeq* the vulnerability-aware binding is almost the same as the baseline. This is due to the characteristic of the DFI design. As shown in Table 5.2, the *diffeq* has much fewer

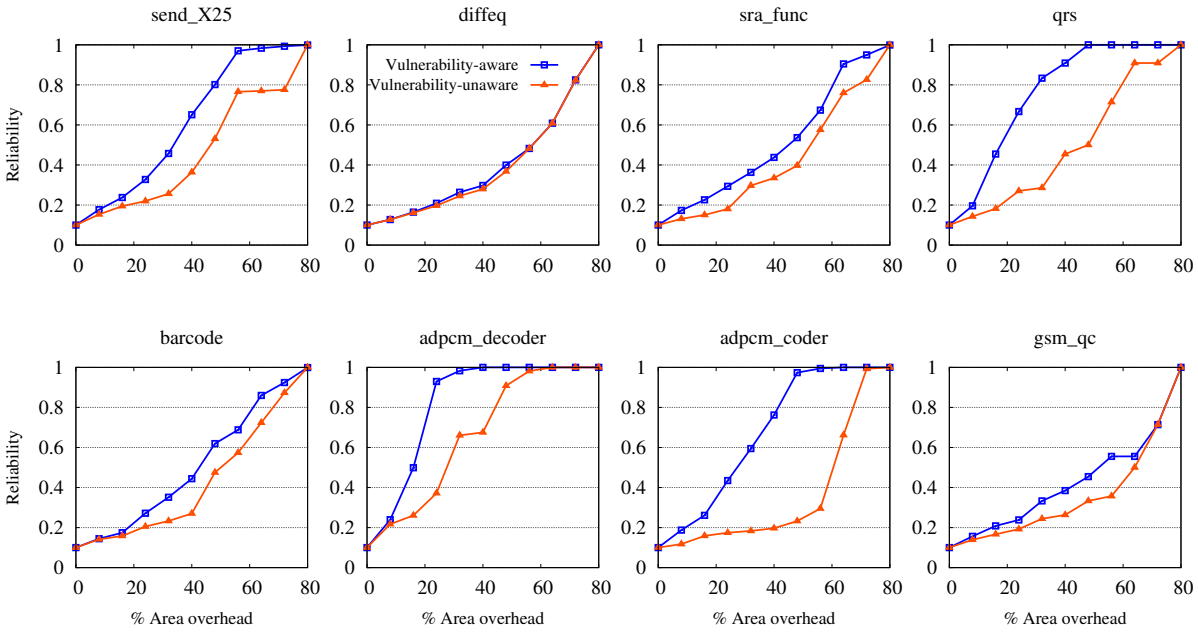


Figure 5.5: The achieved reliability with different level of register protection

conditional branches compared with other CFI benchmarks. In addition, our investigation shows that most of the operations in the DFG of this benchmark are of arithmetic type, which have very low error masking probabilities. Therefore, the variable vulnerability distribution in this benchmark is relatively uniform compared with the CFI benchmarks, and hence the vulnerability-aware binding optimization cannot gain much against the baseline.

To understand the reason for higher achieved reliability, we illustrate different register vulnerability distributions for the scenarios *vulnerability-unaware baseline* and *vulnerability-aware binding* for the benchmark *barcode*, which is also used as the motivation example in Section 5.2. Figure 5.6 shows that for the *vulnerability-unaware baseline* scenario, the vulnerabilities of the registers sorted based on the number of bound variables are not in a decreasing order. While for the *vulnerability-aware binding* scenario, not only the vulnerabilities are highly concentrated in the first few registers, but also the vulnerabilities of all the registers are arranged in a monotonically decreasing order. Therefore, whenever the top registers in this order are chosen to be protected, they always yield the maximum coverage to assure the most efficient error mitigation.

Table 5.2 quantitatively shows the protection efficiency improvement of the proposed vulnerability-aware binding in contrast to the baseline. We can see that with the same level of register protection (i.e. hardware overhead), the proposed vulnerability-aware binding can reach up to 89% reliability improvement. Regarding runtime, the model checking to evaluate variable vulnerabilities takes several minutes, while the ILP optimization on both the number of registers and vulnerability compaction can be finished in a few seconds.

Table 5.2: Reliability improvement and runtime for different binding scenarios

Benchmark	Characteristics			Runtime (sec.)			Reliability improv. of vulnerability aware binding over baseline		
	Var.	Branch	Loop	Vul. eval.	Min. reg.	Vul. compac.	10% overhead	20% overhead	30% overhead
diffeq	24	2	1	1.4	0.2	0.1	0.0 %	2.0 %	6.0%
send_X25	16	5	2	3.7	0.4	0.1	15.6%	22.3%	49.3%
barcode	35	7	2	12.1	0.6	0.2	2.3 %	10.2%	32.4%
sra_func	38	11	0	10.2	0.5	0.1	32.1%	50.2%	63.0%
adpcm_decoder	52	11	1	39.0	3.2	0.6	10.3%	91.6%	150.0%
adpcm_coder	68	13	1	289.3	5.0	1.0	59.4%	64.6%	149.5%
qrs	121	12	0	175.0	10.7	3.1	38.4%	151.5%	151.0%
gsm_qc	256	48	0	295.2	60.2	9.5	12.5%	24.4%	28.1%
average of CFI	-	-	-	103.2	10.1	1.8	**24.4%**	**59.2%**	**89.0%**

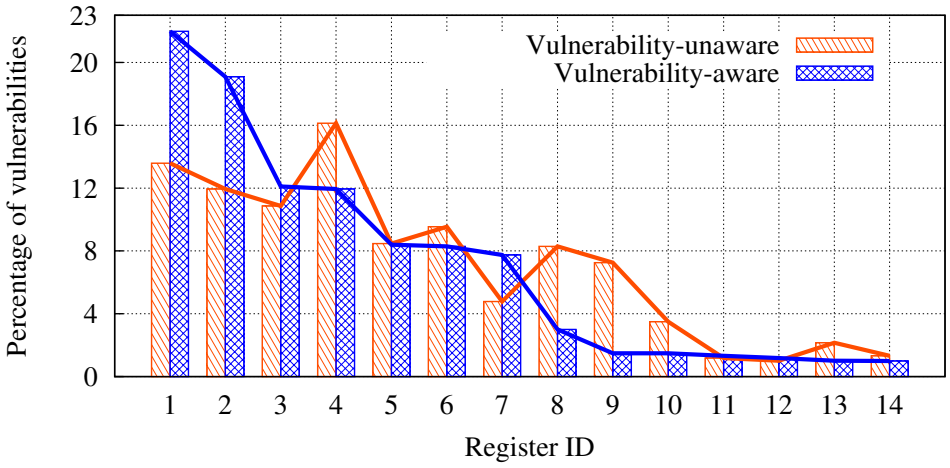


Figure 5.6: Vulnerability distributions of the registers in the vulnerability-unaware baseline and vulnerability-aware binding scenarios for *barcode* benchmark

Table 5.3: ILP runtime, reliability values with different area budgets for the CFI benchmarks

Benchmark	OPs	ILP Runtime (sec.)			Reliability values with different area budgets		
		min. area	min. SFR	area-constr.	10% overhead	20% overhead	30% overhead
send_X25	14	0.1	0.1	0.3	0.80	0.91	0.97
barcode	22	0.2	0.1	1.0	0.76	0.89	0.94
sra_func	35	0.4	0.2	6.5	0.76	0.86	0.94
adpcm_decoder	37	0.5	0.1	7.1	0.83	0.92	0.98
adpcm_coder	48	1.2	0.2	11.9	0.88	0.98	0.99
qrs	88	7.9	1.0	61.8	0.97	0.98	0.99
gsm_qc	101	10.9	0.4	89.8	0.95	0.96	0.98
Average	-	3.0	0.4	25.5	**0.85**	**0.93**	**0.97**

Table 5.4: Comparison of the heuristic binding with ILP-based binding

Benchmark	Heuristic reliability with different area budgets			Deviation from the optimum ILP values			Heuristic runtime (s)	Speedup
	10% overhead	20% overhead	30% overhead	10% overhead	20% overhead	30% overhead		
send_X25	0.76	0.84	0.91	5.2%	8.4 %	6.1%	0.04	8 X
barcode	0.73	0.79	0.87	4.6%	11.4%	7.1%	0.09	73 X
sra_func	0.75	0.83	0.94	1.6%	3.9 %	0.8%	0.20	36 X
adpcm_decoder	0.76	0.91	0.98	9.5%	1.5 %	0.8%	0.21	57 X
adpcm_coder	0.88	0.98	0.99	1.4%	0.1 %	0.2%	0.30	203 X
qrs	0.96	0.98	0.99	1.7%	0.4 %	1.1%	1.03	87 X
gsm_qc	0.94	0.96	0.96	1.3%	0.1 %	1.7%	1.29	69 X
Average	0.82	0.90	0.95	**3.6%**	**3.7%**	**2.5%**	0.45	**76 X**

5.6.4 FU Binding Results Analysis

After register binding analysis, for FU binding we first evaluated our proposed method, and then compared it with the related work without vulnerability consideration.

Table 5.3 shows the runtime of our ILP optimization and the achieved reliability with different area overheads compared with $Cost_{min}$. As ILP-based resource binding is generally an NP-complete problem, the heuristic binding algorithm is proposed in Section 5.5 and here its accuracy and run-time efficiency are also evaluated.

Accuracy and Efficiency of Heuristic Binding

As shown in Table 5.4, on average the heuristic binding can generate the RTL designs with reliability values around 4% deviation from the optimal ILP results, while achieving 76X speedup. Figure 5.7 shows how the runtime increases with the number of behavioral operations, and we can see that the trend of the ILP-based binding is close to exponential, while the heuristic runtime is close to linear. This is because the used RTL library is fixed, i.e. the number of FU versions V is constant. According to the analysis $O(N \cdot Vlog(N \cdot V))$ in Section 5.5, the runtime complexity becomes $O(N \cdot logN)$.

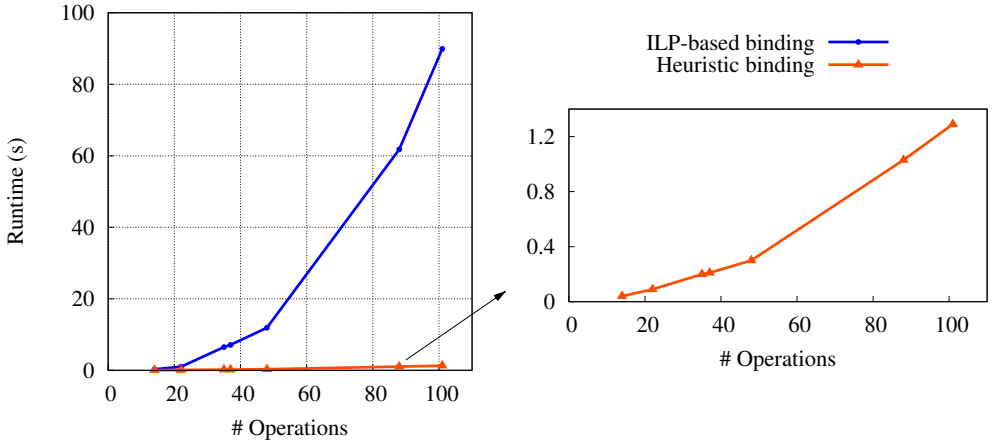


Figure 5.7: Comparison on the runtime of the ILP-based and heuristic binding

Comparison with Related Work

To show the importance of behavioral vulnerabilities, we compare our vulnerability-aware FU binding with the binding technique in [81]. In this related work only the reliability and area metrics of RTL components are known, and it starts from the initial solution which employs the most reliable version for each bound FU (i.e. the initial binding has the minimum failure rate SFR_{min} in our case). If the overall binding area is greater than the user-specified constraint, a victim RTL resource is selected based on its area to be replaced with a smaller version. This procedure is repeated until the area constraint is met or all FUs use the least reliable versions.

Figure 5.8 compares the achieved reliability for both our vulnerability-aware binding and the vulnerability-unaware binding in [81]. It is clear that with the same amount of additional area starting from minimum value, the vulnerability-aware binding generates a system with much higher reliability. Guided by behavioral vulnerability information, the ILP optimization can selectively harden the RTL FUs bound with the most vulnerable operations. While for the vulnerability-unaware binding, the FUs are hardened according to the priority determined by their RTL area value. This priority is very likely to be non-optimal, as FUs with small area do not necessarily bind with vulnerable operations. Therefore the reliability improvement with area increase is slower than in the vulnerability-aware case. Note that for the benchmark *gsm_qc*, both binding have almost the same reliability values. This is because the area and failure rate of RTL FUs are discrete values, and furthermore, this benchmark uses RTL multipliers, which have dominant area compared with other smaller FUs such as adders, comparators. Therefore, the absolute area of 10% interval is rather large. Within the first 10% area, almost all the small FUs are hardened and only multipliers are left for further hardening. In this regard, the vulnerability-aware binding is more suitable and efficient for the scenarios that absolute area budget is limited and fine-grained binding space exploration is essential, such as the cases for other benchmarks in Figure 5.8. On average, compared with the binding in [81] our vulnerability-aware binding can reach 25% relative higher reliability with the same area budget.

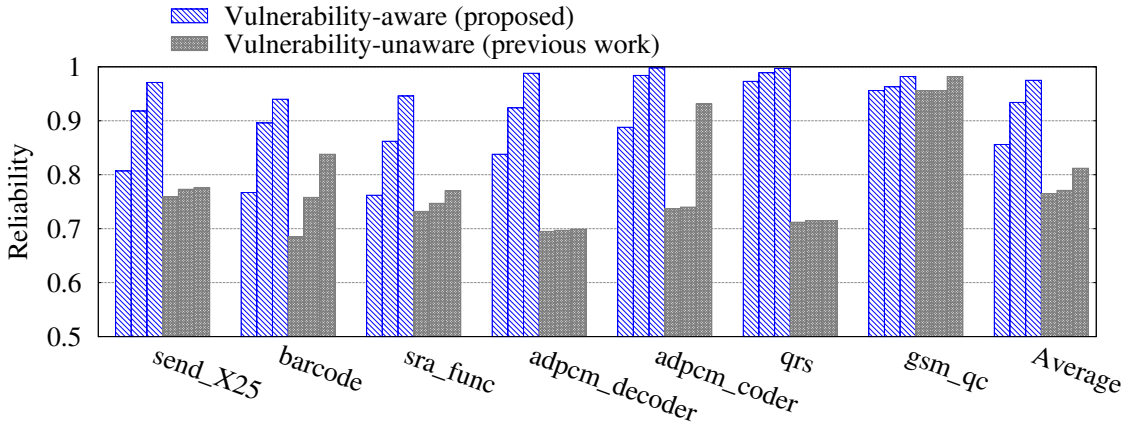


Figure 5.8: Comparison between our vulnerability-aware binding and the vulnerability-unaware binding with the sequence of 10%, 20% and 30% area overhead

5.6.5 Combined analysis of register and FU binding

To investigate the overall reliability improvement considering the register and FU binding together, we performed the combined analysis.

Combined reliability analysis

In Table 5.5, the FIT rate contributions of registers and FUs are computed for each benchmark. We can see that on average the RTL registers contribute to a large portion (around 88%) of the overall SER, which is in the similar range as reported in [36].

In addition, when register and FU binding are considered together, on average our vulnerability-aware binding can reach up to 85% reliability improvement compared with the vulnerability-unaware scenarios. The improvement ratio is close to the pure register binding case in Table 5.2 due to two main reasons. First, the register SER contributions are higher than that of FUs to the entire system. Second, according to Table 5.1, the DICE register hardening has around 10X FIT reduction while FU hardening has only around 1.5X FIT reduction with the same additional area. Nevertheless, with smaller technology nodes the SER contributions of combinational circuits will increase fast and become even comparable to register SER [161], therefore the impact of selective FU hardening will become more profound.

Validation by RTL Fault Injection

To validate our behavioral vulnerability analysis and the following reliability-aware binding technique, we also performed Fault Injection (FI) in the generated RTL implementations of the benchmarks. First we used LegUp to generate the corresponding RTL Verilog description. Then the RTL design is running with input stimulus from the benchmark suits [158] and the bit flip is injected at the output of random FU module and random clock cycle. For each injection, we continue running the applications until the error-free execution is finished, then the functional output of the error-free and error-injected design are compared to determine whether there exists system failure. According to different binding solutions, the RTL components can

Table 5.5: Combined reliability improvement of vulnerability-aware binding over the vulnerability-unaware case with the same area budgets

Benchmark	FIT rate contribution		Reliability improvement over vulnerability unaware binding		
	Register	FU	10% overhead	20% overhead	30% overhead
send_X25	85.2%	14.8%	15.3%	22.3%	48.1%
barcode	80.6%	19.4%	2.1%	10.5%	31.1%
sra_func	85.5%	14.5%	31.5%	49.2%	61.5%
adpcm_decoder	90.0%	10.0%	9.6%	89.0%	140.0%
adpcm_coder	89.0%	11.0%	58.8%	63.7%	142.3%
qrs	96.2%	3.8%	38.4%	149.4%	148.1%
gsm_qc	89.6%	10.4%	12.2%	23.8%	27.3%
Average	88.0%	12.0%	**24.0%**	**57.2%**	**85.5%**

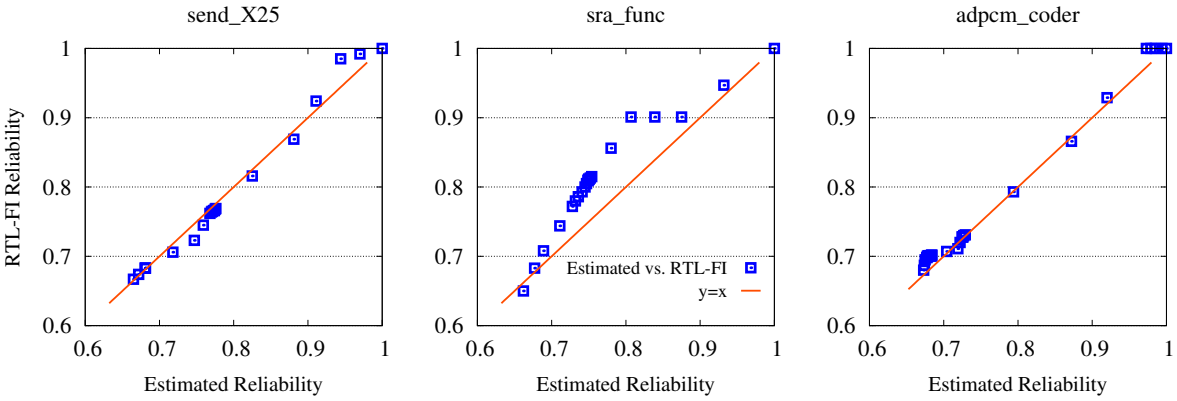


Figure 5.9: Comparison of the reliability values from our behavioral estimation and RTL FI

have different SER (i.e. FIT), which are used to calculate the system reliability.

To illustrate the consistency between RTL FI reliability and the estimated values using behavioral vulnerability analysis in HLS, we evaluate three benchmarks as representatives, and show the comparison in Figure 5.9. From this figure we can see that the estimated reliability with behavioral vulnerability and the real system reliability from RTL FI are close to each other (on average with 3% and maximum with 10% difference), therefore can validate the results of our reliability-aware binding technique.

5.7 Conclusion and Summary

Performing soft error mitigation during the HLS process is necessary for cost-efficient reliability improvement. Sequential and combinational elements, i.e. registers and FUs in the HLS flow, are the main contributors to the system soft error rate, when the memory blocks are efficiently protected with ECC codes. This chapter introduced a approach to explore reliability-aware register and FU binding with comprehensive analysis of the variable and operation vulnerabilities. For the vulnerability evaluation, we used a state-based method to probabilistically

investigate the soft error propagation and masking in the control and data flow graph. Then based on the obtained vulnerability values, both ILP-based and heuristic optimization are performed to bind vulnerable registers/operations to hardened registers/FUs using the selective protective scheme. The experimental results show that the proposed binding techniques reach much more efficient reliability enhancement than the vulnerability-unaware binding, i.e. up to 85% reliability improvement with the same area budget.

Generally speaking, the reliability-aware optimization on resource allocation and binding are in the spatial domain during the HLS process. In the next chapter, we will discuss how to achieve similar optimization in the temporal domain, i.e. handling reliability-aware scheduling in HLS.

6 Reliability-aware Operation Chaining

In previous chapter the reliability enhancement potentials are explored during the allocation and binding phases of high level synthesis, i.e. in the *spatial* domain. In this chapter we will introduce a novel scheduling approach in the *temporal* domain to further investigate these potentials in high level synthesis. Inspired by the observation that the timing resource within individual clock cycle can be redistributed to maximize reliability, we propose a reliability-aware operation chaining technique, considering both the behavioral operation vulnerabilities and RTL reliability-cost trade-offs in functional units.

6.1 Introduction

In addition to the allocation and binding steps introduced in last chapter, *scheduling* is another important step in the HLS process. Scheduling plays a critical role in exploring the concurrent execution to improve the performance of the generated RTL design [162]. As there exist large delay variations for executing different behavioral operations, for the performance improvement it is possible to chain multiple operations bound to fast FUs in a single clock cycle (i.e. *operation chaining*) or execute an operation bound to slow FU across multiple clock cycles (i.e. *multicycling*).

In the scope of reliability aware HLS, there have been previous attempts to investigate the reliability potentials of scheduling and binding [24, 81, 86, 163, 164]. The work in [85, 86] explored the expensive duplication or triple modular redundancy of FUs, while the authors in [81, 163, 164] investigated the reliability potentials of using multiple implementations of FUs. They focused only on the multicycling techniques in the scheduling of DFI designs. As discussed in Chapter 5, the DFI designs are characterized by significant amount of arithmetic operations (e.g. addition, multiplication and division) with few control dependencies. Due to the long execution delays of such operations, multicycling scheduling technique is mainly adopted and investigated. In contrast, the CFI designs contain many relational and logic operations (e.g. comparison, logic *AND* operations) which typically have short execution delays. Therefore, operation chaining is more appropriate scheduling technique for CFI designs. Recent fault injection results [165, 166] showed that control flows are more vulnerable to the transient errors than data flows, so the reliability enhancement of the operations in the CFI designs is a very critical task. Actually, due to unbalanced branch execution probabilities and error masking effects in CFI designs [24], the operations in these designs are likely to manifest non-uniform vulnerabilities. Hence, selective enhancement of the operations becomes a promising solution for cost efficient reliability improvement. However, these potentials are not explored in the literature yet.

In this chapter we propose a *reliability-aware operation chaining* technique to achieve efficient reliability enhancement in the CFI designs. This technique redistributes the timing

resource during the chaining process from the reliability perspective, i.e. within given timing constraints, “borrow" time from the robust operations and use it for hardening vulnerable ones. Our scheduling and binding formulation takes into consideration not only the operation vulnerabilities at the behavioral level, but also the reliability and delay characteristics of RTL components. Compared to the traditional operation chaining, experimental results show that with the same timing constraint, our proposed technique can reduce the system failure rates by 3X with only 15% area and 16% power overhead.

The rest of this chapter is organized as follows. Section 6.2 introduces the HLS preliminaries and motivates the proposed work. Section 6.3 describes both the ILP formulation and heuristic algorithm of the reliability-aware operation chaining. In Section 6.4 the experimental results are discussed and finally Section 6.5 concludes the chapter.

6.2 Preliminaries and Motivation

In the scheduling phase of HLS, the behavioral operations are partitioned into different groups and all the operations in the same group can be concurrently executed. Many techniques have been investigated to improve the design performance, such as multicycling and operation chaining [39, 40, 167]. The multicycling technique reduces the clock period by executing a slow operation across multiple clock cycles with either a non-pipelined or pipelined FU. Operation chaining is performed when the dependent operations can satisfy the following requirement [167]: Two operations i and j that have a data dependency between them can be chained, if their execution time d_i and d_j are such that $d_i + d_j \leq clk\_period$, where clk_period is the allocated clock period for scheduling the design. Also, operation j can start execution only after the execution of i has finished.

In the scope of reliability-aware scheduling, multicycling technique has been well researched in previous work targeting DFI applications [81, 163, 164]. In contrast, our work focuses on reliability-aware operation chaining in CFI designs. The basic idea is to redistribute the time within clock cycles among the dependent operations according to their criticality, i.e. assign less time to execute non-vulnerable operations (using fast but less reliable FUs) and more time to harden the vulnerable operations, so that the overall reliability of the generated RTL can be maximized.

6.2.1 Behavioral Design and Reliability Models

In our work the behavioral design is represented as a Control and Data Flow Graph (CDFG), which is a directed graph consisting of two level structures: control flow graph and Basic Blocks (BBs), as shown in Figure 6.1. Each BB includes a sequence of operations, and there exist control dependencies between BBs. To incorporate the behavioral reliability information into the HLS process, our work uses the metric *operation vulnerability*, which is defined as the probability of system failure given errors occur during the execution of this operation. Therefore it is a conditional probability which reflects the error sensitivity of the behavioral design to each individual operation.

For the vulnerability evaluation, we adopt the *single fault model*, i.e. during the entire design execution, fault occurs at only one operation. Actually, according to the failure rates

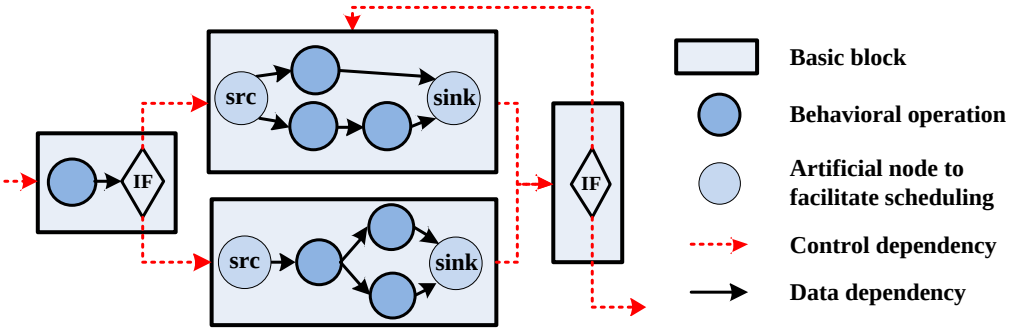


Figure 6.1: An example of control and data flow graph

reported in [53], the probability of multiple faults occurring spatially or temporally in different hardware modules is extremely low. In addition, we consider the aggregated fault effects as erroneous value at the operation output, so various fault types (e.g. single/multiple faults within the same hardware module) can be modeled as well.

We use our previous model checking technique presented in Section 4.5 to obtain the operation vulnerabilities. For error generation modeling, the impaired operation receives correct inputs but generates erroneous output. For error propagation and masking, the CDFG is modeled as a state transition system, which is then analyzed by probabilistic model checker. Note that the obtained vulnerability is independent of the schedules and the final RTL implementations, but only determined by the error propagation properties, i.e. the functionalities of the behavioral design.

6.2.2 Reliability-aware Chaining in CFI Designs

To integrate reliability into the scheduling and binding steps, there are two essential factors to be considered: the failure rate of an RTL FU and the vulnerability of a behavioral operation. The *failure rate* of an FU determines the error occurrence rates, which is a reliability measurement of that FU, while the *vulnerability* of a behavioral operation shows the probability that the occurred errors propagate and finally appear in the design outputs as visible failures.

During HLS, the hardware components (e.g. registers and FUs) implementing the variables and functions in the behavioral model, are selected from an RTL library [39]. In this resource/component library, there can be various implementations of the same FU type with different reliability characteristics [81]. As a running example, Figure 6.2(b) shows the delays and failure rates of the FUs in an RTL component library, which are characterized regarding soft errors using the approach detailed in Section 6.4.2. The unit of failure rate is FIT (Failure In Time), i.e. 1 FIT is equal to 1 error per 10^9 hours.

In the behavioral CFI designs, due to the unbalanced branch executions and error masking effects, the operations are very likely to have non-uniform vulnerabilities. For instance, in Figure 6.2(a) the vulnerability of OR operation is evaluated as 0.5 due to the error masking effect of the operation $>$, while others are 1.0 because the outputs are directly affected by the errors.

Due to these two factors, for the generated RTLs with different FU implementations, the contribution of each individual operation to the overall system failures can be significantly

different. To better illustrate this phenomenon, the running example are investigated with the number of *Failures per Execution* (FPE) as the reliability metric:

$$FPE = \sum_{i=1}^{N_{OP}} \sum_{v=1}^{N_{FU}} Vul(OP_i) \cdot active\_period(OP_i) \cdot b_{iv} \cdot Failure\_rate(FU_v)$$

where N_{OP}, N_{FU} are the total number of behavioral operations and FU versions, respectively. $active\_period(OP_i)$ is the timing duration that OP_i is active, and b_{iv} represents the binding relation between operation and FU. Note that each operation can only be bound to one FU version for each schedule. The following two cases are investigated:

Case I : The scheduling takes two cycles by chaining the operations with nominal FUs (i.e. reliability-unaware);

Case II : Considering the behavioral and RTL reliability information, the cycle number is the same but the scheduling and binding solution is different.

Note that due to clock period limitation, in case II operation > is bound to hardened FU but operation *OR* still uses the nominal version. This is because considering both the operation vulnerabilities and FU failure rates, more failure reduction can be achieved by hardening > rather than *OR*. The detailed comparison of the two schedules is shown in Figure 6.2(c). It can be seen that compared with reliability-unaware schedule I, by making full use of the timing slacks and selectively hardening the operations, schedule II can reduce the FPE by 2.5X with only 7% area overhead.

6.3 Reliability-aware Scheduling and Binding

With the available behavioral vulnerabilities and characterized RTL component library, the proposed formulation of reliability-aware scheduling and binding is described as follows.

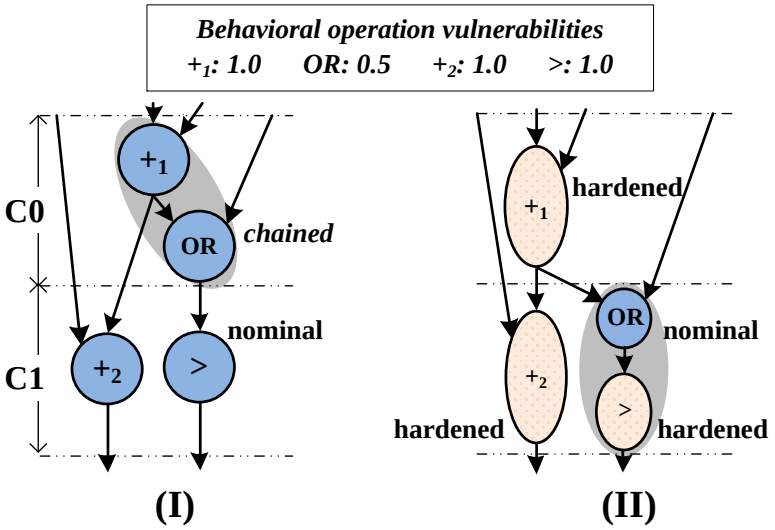
Given: 1) a CDFG with operations as graph nodes and dependencies as graph edges; 2) operation vulnerabilities; 3) an RTL component library of FUs with different delays, areas and failure rates; 4) the allocated clock period; 5) user specified constraints, such as timing, or resource constraint for the generated RTL implementation.

Objective: Schedule and bind the behavioral operations with different FUs, such that the FPE of the generated RTL is minimized, while the user specified constraints are satisfied.

We leverage the ILP technique in [162] to facilitate the modeling of basic scheduling constraints. As it can only handle *fixed* operation delay with single version of FU, our work introduces the new formulation of reliability-aware operation chaining with *variant* delays, i.e. multiple versions of FUs.

6.3.1 Definitions and Notations

The notations used in our reliability-aware scheduling are listed in Table 6.1. For each operation i in the CDFG, there is one FU type k to execute this operation, e.g. a behavioral + operation is to be executed by an RTL adder. There are several constant metrics of the behavioral operations and RTL FUs, and furthermore, two integer scheduling variables $start_i$ and end_i determine the starting and ending clock cycle of operation i. For a single cycle operation



(a) Two chaining cases with 1.2 ns clock period: I) with nominal FUs; II) reliability-aware with hardened FUs

FU version	Delay (ns)			Failure Rate ($\times 10^{-3}$ FIT)			Area (μm^2)		
	+	OR	>	+	OR	>	+	OR	>
0 (nominal)	1.0	0.2	0.7	6	2.1	1.5	420	50	110
1 (hardened)	1.1	0.4	0.9	1.8	1.6	1.1	440	70	130

(b) Delay, area and reliability of an RTL component library

Schedules	FPE ($\times 10^{-24}$)	Total Delay (cycles)	Area (μm^2)
I	5.8	2	580
II	2.3	2	620

(c) Metrics of two different schedules

Figure 6.2: Running example of reliability-aware chaining

$end_i = start_i$, while for multi-cycle operation $end_i > start_i$. The binding variables b_{iv} determine which FU version v will be used for the execution of operation i, and the corresponding FU delay will in turn influence the scheduling of dependent operations.

6.3.2 Basic Scheduling Constraints

Data Dependency

For each data dependency edge from operation i to j, the following constraint is to make sure that operation j only starts after the execution of operation i:

$$end_i \leq start_j \tag{6.1}$$

Control Dependency

Control dependencies exist between the BBs of CDFG, and each BB is polarized with two artificial nodes: super source *src* and super sink *sink* as shown in Figure 6.1. For each control dependency from basic block BB_i to BB_j the following constraint is added

$$end_{sink_{BB_i}} \leq start_{src_{BB_j}} \tag{6.2}$$

Timing Constraint

This constraint guarantees that the user-specified timing requirement, maximum L number of cycles for the *longest path latency* [162], will be met for the exit basic block BB_{exit} in the CDFG

$$end_{sink_{BB_{exit}}} \leq L \tag{6.3}$$

Resource Constraint

Each operation should be implemented by one and only one version of the FU

$$\sum_{v=1}^{FU_k} b_{iv} = 1 \text{ for each operation } i \tag{6.4}$$

Note that two concurrent operations in the same cycle cannot be executed by the same FU instance. Therefore, we need to find the maximum number n_{vk} of instances of the version v for FU type k:

$$n_{vk} \geq \sum_{i \text{ executed by FU type } k} x_{icv} \text{ for each clock cycle } c$$

Then the area/power constraints are formulated as

$$\sum_{k=1}^{M} \sum_{v=1}^{FU_k} a_{vk} \cdot n_{vk} \leq A, \quad \sum_{k=1}^{M} \sum_{v=1}^{FU_k} p_{vk} \cdot n_{vk} \leq P \tag{6.5}$$

Table 6.1: Notations for reliability-aware scheduling

Notation	Definition	Type
N	Number of operations to be scheduled; operation i is executed by specific FU type k, $k \in [1, M]$	Constant
M	Number of FU types	Constant
FU_k	Number of implementation versions of FU type k	Constant
Vul_i	Behavioral vulnerability of operation i	Constant
L	Timing constraint (number of cycles) on the schedule	Constant
A	Area constraint on the schedule	Constant
P	Power constraint on the schedule	Constant
a_{vk}	Area of the version v of FU type k	Constant
p_{vk}	Power of the version v of FU type k	Constant
λ_{iv}	Failure rate of operation i executed by version v of FU type k	Constant
d_{iv}	Delay of operation i executed by version v of FU type k	Constant
x_{icv}	$x_{icv} = 1$ if operation i is scheduled in cycle c and executed by version v of the FU. $i \in [1, N], c \in [1, L], v \in [1, FU_k]$	Binary variable
b_{iv}	$b_{iv} = 1$ if operation i is executed by version v of the FU. $b_{iv} = \sum_c x_{icv}, i \in [1, N], v \in [1, FU_k]$	Binary variable
$start_i$	Start clock cycle of operation i, $start_i = \sum_c \sum_v c \cdot x_{icv} \in [0, L]$	Integer variable
end_i	End clock cycle of operation i, $end_i \in [0, L]$	Integer variable
d_i	Execution delay of operation i, determined by the used FU version: $d_i = \sum_v b_{iv} d_{iv}$	
$[d_i]$	Delay interval of operation i with lower and upper bounds from the RTL component library, $[d_i] = [d_i^l, d_i^u], d_i \in [d_i], d_i^l = min(d_{iv}), d_i^u = max(d_{iv}), \forall v \in [1, FU_k]$	

6.3.3 Operation Chaining Constraints

In the conventional chaining, the propagation delay of an operation is fixed with a *single* FU version in the RTL component library, therefore, the overall path delay can be directly computed before scheduling starts [40, 168]. If the path delay from operation i to its nearest successor operation j (e.g. operation $+_1$ and $+_2$ in Figure 6.2) exceeds the clock period, in addition to the dependency constraint in Equation (6.1), another constraint has to be added to separate them into different cycles [168]

$$end_i + 1 \leq start_j \tag{6.6}$$

However, for our reliability-aware operation chaining, there are *multiple* versions of FU implementation. Therefore, the path delay becomes dependent on the binding solutions. Actually the delay of operation i is a function of the binding variables, $d_i = \sum_{v=1}^{FU_k} b_{iv} d_{iv}$. To overcome this inter-dependency between scheduling and binding, we propose to use *interval arithmetic* to find the candidates of operation chaining.

The basic idea is to calculate the path delay intervals based on the operation delay intervals. After comparing these intervals with the clock period, we can determine whether the dependent operations can be chained or must be separated into different cycles. The detailed algorithm for the generation of chaining constraints is listed in Algorithm 6.1. Note that the lower bound d_i^l and upper bound d_i^u for each operation i are extracted from the RTL component library, and are constant values, e.g. $d_{OR}^l = 0.2, d_{OR}^u = 0.4$ in Figure 6.2. We calculate the interval of

path delay from operation i to j with this formula:

$$\begin{cases} [d_{path_{ij}}] = [d_i^l, d_i^u] + [d_j^l, d_j^u] = [d_i^l + d_j^l, d_i^u + d_j^u] \\ d_{path_{ij}} \in [d_{path_{ij}}] = [d_{path_{ij}}^l, d_{path_{ij}}^u] \end{cases} \tag{6.7}$$

Depending on the comparison of clock period with the lower and upper bounds of this path delay interval, three cases are handled differently:

1. $clk\_period < d_{path_{ij}}^l$: The lower delay bound exceeds the clock period. This means for operation i and j, no matter which versions of FU are bound, they must be scheduled into different cycles. Therefore, constraint (6.6) is added;
2. $d_{path_{ij}}^u < clk\_period$: The upper delay bound is smaller than the clock period. This means operation i and j can be safely chained, and the dependency constraint (6.1) is enough;
3. $d_{path_{ij}}^l \leq clk\_period \leq d_{path_{ij}}^u$: Depending on the binding solution, if $clk\_period < d_{path_{ij}}$, then the two operations must be separated with constraint (6.6), otherwise they can be chained.

The modeling difficulty in case 3 arises due to the *conditional* constraints, which can be stated using the following notation

$$\begin{aligned} &f(i,j) = clk\_period - d_{path_{ij}} \\ &g(i,j) = end_i + 1 - start_j \\ &\text{if } f(i,j) < 0, \text{ then } g(i,j) \leq 0 \end{aligned} \tag{6.8}$$

Such kind of conditional constraint cannot be directly modeled by linear programming method, therefore we need to convert it to linear constraints.

The basic idea is to use a 0/1 variable linked to the “condition" to indicate whether it holds or not. Therefore, one additional binary indicator variable y, a sufficiently small lower bound L_f (negative value) on $f(i,j)$ and a sufficiently large upper bound U_g on $g(i,j)$ are introduced. If $f(i,j) < 0$, $y = 1$, otherwise $y = 0$. Then the following two linear constraints are added to model this conditional constraint

$$\begin{cases} f(i,j) \geq L_f y \\ g(i,j) \leq U_g(1-y) \end{cases} \tag{6.9}$$

Testing the different values of this indicator variable y, the above constraints yield:

$$y = 0 \Rightarrow \begin{cases} f(i,j) \geq 0 \\ g(i,j) \leq U_g \end{cases}, \quad y = 1 \Rightarrow \begin{cases} f(i,j) \geq L_f \\ g(i,j) \leq 0 \end{cases} \tag{6.10}$$

We can see that for each value of y, the constraints with U_g or L_f are tautology and the remaining ones just represent the semantics of the conditional constraint. As the lower and upper bounds of the path delay $d_{path_{ij}}$ are already available, the values of the constants L_f, U_g actually can be easily set.

As shown in Algorithm 6.1, for each operation to be scheduled in the CDFG, we perform a modified depth first search along its data dependent paths. During the search process, the real propagation delay of the path is computed as a function of the binding variables (Line 6, 7 and 15, 16). In addition, the delay intervals of the path are also calculated according to the

Algorithm 6.1 Chaining Constraints Generation with Variant Operation Delays

1: **Input**: CDFG of the behavioral function, clock period, different FU implementations in an RTL component library
2: **Output**: Chaining constraints for reliability-aware scheduling
3: **for** each BB in CDFG **do**
4: **for** each operation i in BB **do**
5: $[d_{path_i}] = [d_i] = [d_i^l, d_i^u]$ ▷ Constant bounds
6: $d_i = \sum_{v=1}^{FU_k} b_{iv} d_{iv}$ ▷ Operation delay
7: $d_{path_i} = d_i$ ▷ Path starting from operation i
8: CHAINING_DELAY_UPDATE$(i, [d_{path_i}], d_{path_i})$
9: **end for**
10: **end for**
11:
12: **function** CHAINING_DELAY_UPDATE$(i, [d_{path_i}], d_{path_i})$
13: **for** each successor operation j of operation i **do**
14: $[d_{path_{ij}}] = [d_{path_i}] + [d_j^l, d_j^u]$
15: $d_j = \sum_{v=1}^{FU_k} b_{jv} d_{jv}$
16: $d_{path_{ij}} = d_{path_i} + d_j$
17: **if** $d^l_{path_{ij}} > clk\_period$ **then** ▷ Case 1
18: Add constraint (6.6)
19: **else if** $d^u_{path_{ij}} \geq clk\_period \geq d^l_{path_{ij}}$ **then** ▷ Case 3
20: Add constraint (6.9)
21: **else** ▷ Case 2, further chaining
22: CHAINING_DELAY_UPDATE$(j, [d_{path_{ij}}], d_{path_{ij}})$
23: **end if**
24: **end for**
25: **end function**

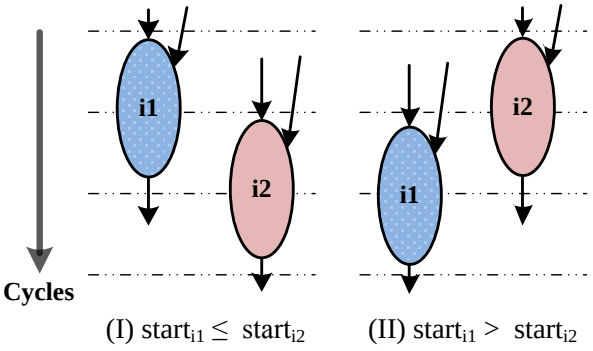


Figure 6.3: Two cases of compatibility extraction from operation scheduling

interval arithmetic (Line 14). By comparing the lower and upper bounds of the path delay interval, we either add the chaining constraints (Line 18 and 20), or search deeper for more chaining potentials (Line 22).

6.3.4 Binding Constraints and Multiplexer Consideration

During the binding process in HLS, the behavioral operations and variables are mapped to RTL FUs and registers, respectively. The reliability-aware register binding is performed using our previous formulation in [24], so here we mainly discuss the operation binding. The key point is how to extract the binding compatibilities of behavioral operations from the above scheduling formulation.

Two different operations $i1$ and $i2$, which have the same operation type (e.g. both are arithmetic additions), are defined as *incompatible* when they are concurrently executed in the

same clock cycle. We use a binary variable $c_{i1,i2} = 0$ to represent this incompatibility. Using the scheduling variables in Table 6.1, the following constraints are used to link the scheduling and binding process:

$$\begin{aligned} & c_{i1,i2} < 1 + OL_{i1,i2} \\ & OL_{i1,i2} = \begin{cases} \frac{start_{i2} - end_{i1}}{L_c}, & \text{if } start_{i1} \leq start_{i2} \\ \frac{start_{i1} - end_{i2}}{L_c}, & \text{else} \end{cases} \\ & L_c > L \Rightarrow -1 < OL_{i1,i2} < 1 \end{aligned} \tag{6.11}$$

where L_c is a constant number larger than the longest path latency L, and $OL_{i1,i2}$ is an auxiliary variable to check if the execution cycles of $i1$ and $i2$ are *OverLapped (OL)*. As shown in Figure 6.3, when the value of $OL_{i1,i2}$ is non-positive, the binary variable $c_{i1,i2}$ is forced to be 0 (i.e. operation $i1$ and $i2$ are incompatible). Similar technique as Equation (6.9) can transform these conditional constraints to linear constraints.

The register and operation bindings determine the multiplexing logics in the generated RTL implementations. Generally, if FU_k has $n_{k,l}$ interconnections at the l^{th} input port from different registers, an $n_{k,l}$-to-1 multiplexer needs to be inserted before this port. In this work we employ the formulation in [156] to model the possible multiplexers. Taking the delay of multiplexer $d_{MUX}(i)$ before the FU executing operation i into consideration, the delay of operation i should be modified as $d_i = \sum_{v=1}^{FU_k} b_{iv} d_{iv} + d_{MUX}(i)$.

6.3.5 Objective

The goal of our reliability-aware operation chaining is to minimize the number of *Failures per Execution* (FPE) of the generated schedule, which takes into consideration the total execution time of the schedule as well. Similar to the running example in Section 6.2.2, for each operation three factors are considered in the computation of FPE: the failure rate of the bound RTL FU, the active period of this operation and its behavioral vulnerability. The first two factors represent the error occurrence, i.e. the number of occurring failures during this operation execution, and the third one represents the error propagation probability, i.e. how much percentage of occurred errors propagate and manifest at the final function output. Note that as explained in Section 6.2.1, the correlations and dependencies in error propagation and masking are already considered by the comprehensive model checking technique. Therefore, the objective of our optimization can be simply formulated as

$$\text{minimize} \quad FPE = \sum_{i=1}^{N} (\sum_{v=1}^{FU_k} b_{iv} \lambda_{iv} \cdot active\_period_{iv}) \cdot Vul_i \tag{6.12}$$

In this formulation, the failure rate λ_{iv} and the $active\_period_{iv}$ of operation i are already known depending on the implemented version v of the executing FU type in the RTL component library. The behavioral vulnerability Vul_i is also evaluated before scheduling, therefore the FPE of the schedule is a linear function of the binding variables b_{iv}.

6.3.6 Heuristic Algorithm

The problem of simultaneous scheduling and binding is known to be NP-hard, and the ILP-based method is generally only applicable to medium size designs [40]. To improve the scala-

Algorithm 6.2 Heuristic Reliability-aware Scheduling and Binding

```
1: All operations are bound to the most reliable FUs
2: Initial schedule with all hardened operations
3: while constraints not met do
4:     Select candidate operations
5:     Calculate the priority metric for each candidate operation
6:     Select the operation with highest priority
7:     Bind a less reliable FU version to this operation
8:     Update the chaining constraints and reschedule the design
9: end while
```

bility, we propose and implement a reliability-aware heuristic algorithm for operation chaining. The basic idea is to prioritize the behavioral operations and schedule the design in an iterative manner. As a guide for the design space exploration, the priority metric will take into consideration both the behavioral and RTL reliability information. Then for each iteration, the reliability-related binding constraints are removed and only conventional scheduling will be performed. Therefore, the runtime of the iterative scheduling can be significantly reduced compared with ILP-based simultaneous binding and scheduling. The detailed description of the heuristic is shown in Algorithm 6.2.

The key point here is to construct the priority function. For FPE minimization, we calculate the priority metric of one operation according to the FPE change when it is bound to different FU versions. Assume operation OP_i can be bound to version v or $v-1$ of the corresponding FU. By binding OP_i to a less reliable version $v-1$ ("softening"), a quick evaluation on the FPE change can be used to compute the priority metric

$$\Delta FPE = FPE_{v-1} - FPE_v, \quad priority(OP_i) = \frac{1}{\Delta FPE}$$

The user-specified constraints can be considered in the selection of *candidate operations* as in Line 4. For instance, if the timing constraint is not met, the operations in the critical paths, rather than all operations in the CDFG, will be chosen as "softening" candidates. Then for the candidate operation with the highest priority, its implementation FU version is modified and the design is rescheduled. This iterative process continues until all constraints are met. If all operations are already implemented by the least reliable FU versions and there are still unmet constraints, the algorithm will report scheduling as infeasible. Note that the priority metric is constructed based on our reliability evaluation, and actually can be integrated with any existing scheduling algorithm in the iteration, such as the scheduling technique with polynomial runtime [162], and other optimization methods [40, 169].

6.4 Experimental Results

The proposed technique is applied to the CDFGs extracted from seven benchmarks, and we use a characterized RTL component library regarding soft errors to demonstrate the advantage of the reliability-aware operation chaining.

6.4.1 Experimental Setup

The used behavioral benchmarks in our experiments are *barcode*, *sra_func*, *send_X25*, *qrs*, *gsm_qc*, and the ADPCM encoder and decoder. The extraction of behavioral CDFGs and the reliability-unaware scheduling [162] are performed with the HLS tool LegUp [153], and we

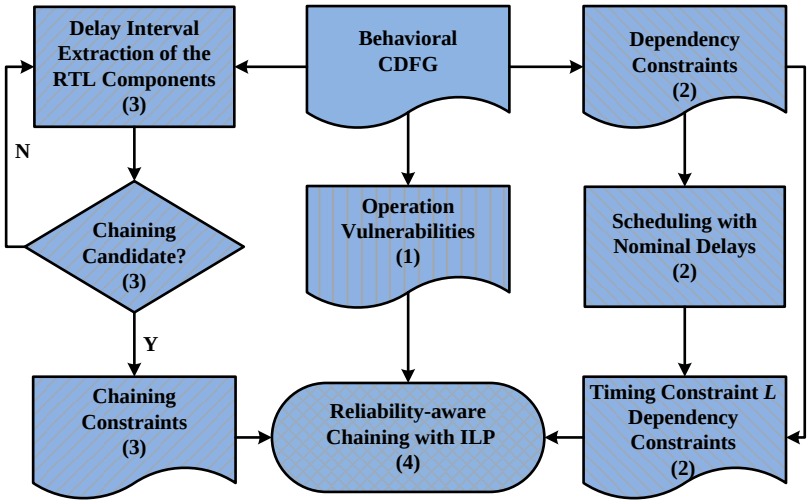


Figure 6.4: Work flow of the proposed operation chaining based on ILP

carry out the ILP optimization using the solver *lp_solve* [170]. The RTL area and power values are reported after logic synthesis by Design Compiler based on the 45 nm Nangate library. The experiments are performed on a workstation with Intel Xeon E5540 2.53 GHz and 16 GB RAM.

As shown in Figure 6.4, the overall work flow of ILP-based operation chaining consists of four main steps: 1) the extracted CDFGs are analyzed to obtain the operation vulnerabilities; 2) the reliability-unaware operation chaining is performed with nominal FU delays, which can give the longest path delay L of the design. This will be used later for timing constraint; 3) characterized RTL component library with several versions of FU implementation is utilized to construct the dependent path delay intervals to generate the chaining constraints as demonstrated in Algorithm 6.1; 4) the ILP optimization is performed to redistribute the operation delays within the timing constraint L to generate the schedule with high reliability.

6.4.2 RTL Component Library Characterization

To explore the delay and reliability trade-off, we need to have various versions of FUs with different delays and SER in the component library. A variety of hardening techniques including gate sizing, voltage scaling, and transient filter insertion were investigated. Transient filters are used as a case study here, as they provide a considerable reduction in SER while imposing

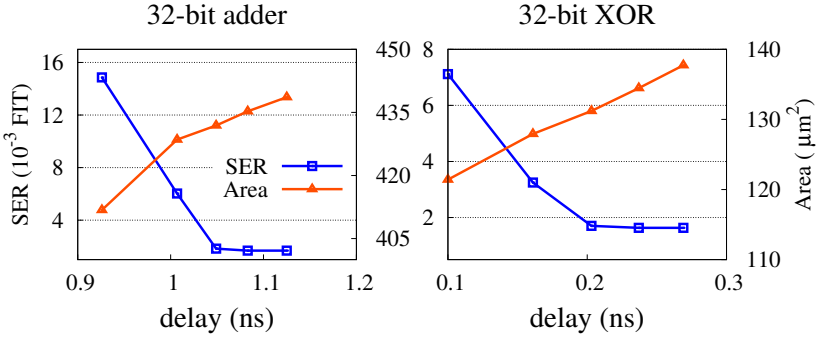


Figure 6.5: Characteristics of two FUs with different hardening levels

Table 6.2: Comparison between our proposed method and the previous techniques

Benchmark	No. OPs	Scheduling without reliability [162]			Proposed scheduling with ILP reliability optimization						Scheduling without vulnerability [81, 163]	
		Area (μm^2)	Power (mW)	FPE (10^{-24})	Area (μm^2)	Power (mW)	FPE (10^{-24})	Area overhead	Power overhead	FPE reduction	FPE (10^{-24})	FPE increase
send_X25	14	1657.1	1.15	6.5	1691.2	1.18	2.6	2.1%	3.0%	2.5 X	5.1	1.9 X
barcode	24	1809.7	1.29	12.6	1839.5	1.32	3.8	1.6%	2.4%	3.4 X	6.3	1.7 X
sra_func	35	2096.2	1.53	10.6	2658.9	1.98	5.6	26.8%	28.7%	1.9 X	15.7	2.8 X
adpcm_decoder	37	3306.1	2.18	7.7	4135.1	2.80	2.3	25.1%	28.6%	3.3 X	14.0	6.0 X
adpcm_coder	48	4405.4	3.16	8.3	5179.5	3.61	3.0	17.6%	14.3%	2.8 X	19.4	6.5 X
qrs	88	6911.6	4.63	7.2	8500.4	5.83	2.4	23.0%	26.0%	3.0 X	23.1	9.5 X
gsm_qc	101	1866.7	1.42	9.8	2094.1	1.57	3.0	12.2%	10.5%	3.2 X	24.5	8.1 X
Average	-	-	-	-	-	-	-	**15.5%**	**16.2%**	**3.0 X**	-	**5.2 X**

small delay and area overheads [4]. To obtain the delay and SER values of FUs with different hardening levels, we performed logic synthesis on the RTL description of each FU to obtain the gate-level netlist, inserted different number of transient filters at each output, and then employed a hierarchical approach for error generation and propagation analysis [53] based on the commercially characterized 45 nm Nangate library. In total we have implemented 5 versions of each FU and use them in the following reliability-aware chaining. The typical delay and FIT curve for the FUs is similar to that in Figure 6.5. Note that the SER reduction saturates when inserting large number of filters, while the area monotonically increases. The clock period is set to 1.2 ns to accommodate one adder and other possible chaining operations.

To better demonstrate the trade-off between the FU delays and reliabilities, for each operation we selected the FU version which is in the middle of the delay-reliability curve as the nominal case. In this way, we have both less reliable and more reliable versions of the FUs in the design space to be explored.

6.4.3 Results Analysis

To show the effectiveness of reliability-aware operation chaining, two different scenarios are considered for comparison:

- *Scheduling without reliability*: similar to case (I) in Figure 6.2(a), only nominal versions of FUs are used in the general scheduling algorithm [162];
- *Proposed scheduling with reliability optimization*: this scenario corresponds to the case (II) in Figure 6.2(a), considering both behavioral and RTL reliability information.

For both scenarios the longest path delay of the schedule is set to be the same, meaning that the investigation does not incur any performance penalty and the proposed approach tries to redistribute the timing resources to improve the design reliability. We perform the comparison on FPE and area/power of the generated schedules. The detailed results are shown in Table 6.2. In addition to FUs, the power and area values for RTL registers and the steering logic like multiplexers are also included here.

We can see that on average the proposed reliability-aware scheduling can reach 3X FPE reduction. This is because it can make full use of the timing slack in each clock cycle, and efficiently redistribute the timing resource among operations with non-uniform vulnerabilities. Hardening FUs intrinsically introduces area and power overhead, and implementing operations of the same type with different versions of FU may also impair the potential of FU sharing. However, due to the reliability-aware selective hardening, our chaining method generates the

Table 6.3: Comparison between the ILP and heuristic binding

Benchmark	No. OPs	ILP		Heuristic			
		FPE ($\times 10^{-24}$)	Runtime (s)	FPE ($\times 10^{-24}$)	Runtime (s)	FPE diff.	Speedup
send_X25	14	2.6	0.03	2.6	0.01	0.0%	3 X
barcode	24	3.8	0.06	4.0	0.01	5.1%	6 X
sra_func	35	5.6	0.29	5.8	0.01	3.9%	29 X
adpcm_decoder	37	2.3	0.13	2.3	0.01	0.0%	13 X
adpcm_coder	48	3.0	15.42	3.0	0.13	0.0%	116 X
qrs	88	2.4	0.39	2.4	0.01	0.0%	67 X
gsm_qc	101	3.0	20.04	3.1	0.23	3.2%	87 X
qrs+gsm_qc	189	-	Time out	6.0	0.43	-	-
gsm_qc+gsm_qc	202	-	Time out	6.8	0.97	-	-
Average	-	-	-	-	-	1.8%	45 X

schedules with rather small area and power overheads, i.e. 15.5% and 16.2% respectively. To evaluate the efficiency of proposed heuristic scheduling, we also compare it with the optimal ILP, as shown in Table 6.3. To investigate the scalability of both methods, we construct two additional applications by connecting the largest benchmarks *qrs* and *gsm_qc*. Due to the rapid increase of the number of variables and constraints, ILP is time out (> 0.5 hour) for the two largest benchmarks, while the heuristic still works well. This speedup is mainly due to the fact that the heuristic leverages the reliability information with different FU implementations to guide an iterative scheduling loop, instead of direct scheduling and binding with all FU versions. For the benchmarks where ILP results are available, the comparison shows that the heuristic generates the schedules with similar FPEs (on average around 2% difference with 45 X speedup).

6.4.4 Comparison with Related Work

In the category of reliability-aware scheduling and binding techniques, the previous work [81, 163] utilized multiple versions of FUs as well, but they assumed that the reliability of one operation is directly determined by that of bound FU, while the behavioral vulnerability information is ignored. Without these operation vulnerabilities, a conservative assumption has to be made that each failure occurring in the FU results in errors at behavioral function outputs, i.e. operation vulnerabilities are 1.0. Using these pessimistic vulnerabilities, on average the FPE of the generated schedule is increased by around 5X, as shown in Table 6.2. We can also see that the FPE increase ratios generally grow with the number of operations in the behavioral designs. This is because in larger designs there are more error masking effects and the operation vulnerabilities are likely to be much smaller than 1.0. This significant difference, which can lead to over-designed RTL with unnecessary resource overhead, shows that behavioral operation vulnerability is an indispensable part for cost-efficient reliability-aware HLS.

6.5 Conclusion and Summary

Complementary to the spatial domain reliability exploration in Chapter 5, in this chapter we proposed a temporal domain optimization techniques, i.e. reliability-aware scheduling, by investigating the operation chaining potential for reliability enhancement. Both the vulnerabilities of behavioral operations and reliability metrics of RTL FUs are considered during the optimization step of operation scheduling. The experimental results show that compared with reliability-unaware chaining, the proposed technique generated the schedule with 3X reduction of the number of failures per execution, while having small area and power overheads.

7 Concluding Remarks and Outlook

Over the past several decades, aggressive device downscaling has enabled exponential increase of the number of computing system in our society. The continuous increasing frequency of circuit operation and reduction in operating voltages have exacerbate the reliability challenges in future technology nodes. Radiation-induced soft errors, as one of the major reliability issues in nanoscale era, have to be carefully taken into consideration as an additional design metric or constraint in the design space exploration.

The state-of-the-art work on soft error evaluation and mitigation was mostly focused at the low abstraction levels. However, the low level solutions are very costly, because they stay at the end of the design flows and the useful high level application semantics are totally ignored. In contrast, at high abstract levels there are significant potentials for cost-effective solutions, because the reliability metric are considering earlier in the design phase. To handle the increasing complexity of future system and leverage the large flexibility in the design space exploration, this thesis deals with the problem of soft error evaluation and mitigation at higher abstraction levels.

For soft error evaluation, we started from logic level and considered the error correlation during their propagation using an accurate and efficient analytical technique. Addressing this correlation is essential for accurate soft error rate estimation, and more importantly, for the cross-layer error abstraction, e.g. from bit errors at logic level to word errors at register-transfer level. Then the abstraction level is raised to RTL, where the data paths and control paths are analyzed using both analytical and formal methods according to their different error propagation properties. These techniques can successfully quantify the non-uniform vulnerability distributions among the RTL registers under different running workloads. Finally, the error occurrence and evaluation are investigated at the even higher level regarding the behavioral functionality. The vulnerabilities of the behavioral variable and operations, i.e. error sensitivities, are analyzed in a comprehensive way using model checking techniques.

Based on the detailed error rate evaluation at high abstraction levels, cost-efficient error mitigation by selective protection or hardening are investigated during the high level synthesis flow. The optimization starting from specifications at higher levels of abstraction can obtain more benefits, because it leverages the behavioral semantics which are ignored by low level soft error mitigation techniques, and can provide larger flexibility in reliability-aware design space exploration. We developed reliability-aware optimization techniques regarding both resource allocation and binding (in the spatial domain), and operation scheduling (in the temporal domain). The soft error vulnerabilities of behavioral variables and operations are carefully taken into consideration. Compared with the state-of-the-art techniques, our proposed technique can achieve much more efficient reliability enhancement.

Following the famous Moore's law, the International Technology Roadmap for Semiconductors [32] predicts a drastic reduction of both the transistor supply voltage and physical gate

size. According to this roadmap, in the next 10 years the number of transistors per system-on-chip (SoC) will increased by a factor of 12. This direction for further progress is labelled *More Moore* [28]. Concerning this More Moore trend, not only radiation-induced soft error, but also other reliability challenges caused by manufacturing and environmental imperfections, variabilities and threats, e.g. voltage droop, transistor aging and electro migration [33, 171] are already coming into scope.

This thesis focuses on the soft error issue and proposes various evaluation and mitigation techniques. Nevertheless, the entire framework is not restricted to this specific fault model. The error propagation analysis in the circuit network as well as the general reliability-aware optimization algorithms are all suitable to integrate other fault models, such as voltage droop and transistor aging. How to identify the difference between error occurrence models, and adjust our framework to handle different reliability-aware scenarios, are very interesting and promising directions in the future work.

Bibliography

[1] K. Rodbell and C. Cher, "Where radiation effects in emerging technologies really matter," in *IEEE NSREC Short Course*, 2013.

[2] P. Eles, K. Kuchcinski, and Z. Peng, *System Synthesis with VHDL*. Springer, 2010.

[3] S. Mukherjee, *Architecture Design for Soft Errors*. Morgan Kaufmann, 2008.

[4] M. Choudhury, Q. Zhou, and K. Mohanram, "Soft error rate reduction using circuit optimization and transient filter insertion," *Journal of Electronic Testing*, vol. 25, no. 2-3, pp. 197–207, 2009.

[5] S. Z. Shazli and M. B. Tahoori, "Using boolean satisfiability for computing soft error rates in early design stages," *Microelectronics Reliability*, vol. 50, no. 1, pp. 149–159, 2010.

[6] S. Seshia, W. Li, and S. Mitra, "Verification-Guided Soft Error Resilience," in *Design, Automation and Test in Europe*, pp. 1–6, 2007.

[7] G. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, no. 8, Apr. 1965.

[8] S. Mukherjee, J. Emer, and S. Reinhardt, "The soft error problem: an architectural perspective," in *International Symposium on High-Performance Computer Architecture*, pp. 243–247, 2005.

[9] M. Nicolaidis, *Soft Errors in Modern Electronic Systems*, ser. Frontiers in Electronic Testing. Springer US, 2010.

[10] T. C. May and M. H. Woods, "A new physical mechanism for soft errors in dynamic memories," in *International Reliability Physics Symposium*, pp. 33–40, April 1978.

[11] J. F. Ziegler and W. A. Lanford, "Effect of Cosmic Rays on Computer Memories," in *Science*, vol. 206, no. 4420, pp. 776–788, 1979.

[12] J. F. Ziegler and H. Puchner, *SER–History, Trends and Challenges: A Guide for Designing with Memory ICs*. Cypress, 2004.

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

[14] R. Baumann, "Soft errors in advanced computer systems," *IEEE Design and Test of Computers*, vol. 22, no. 3, pp. 258–266, 2005.

[15] E. Ibe, H. Taniguchi, Y. Yahagi, K. Shimbo, and T. Toba, "Impact of scaling on neutron-induced soft error in srams from a 250 nm to a 22 nm design rule," *IEEE Transaction on Electron Devices*, vol. 57, no. 7, 2010.

[16] S. Michalak, K. Harris, N. Hengartner, B. Takala, and S. Wender, "Predicting the number of fatal soft errors in los alamos national laboratory's asc q supercomputer," *IEEE Transactions on Device and Materials Reliability*, vol. 5, no. 3, pp. 329–335, Sept 2005.

[17] "Sun screen," Forbes Magazine, http://members.forbes.com/global/2000/1113/0323026a.html, Nov. 2000.

[18] D. Gajski and R. Kuhn, "Guest Editors' Introduction: New VLSI Tools," *Computer*, vol. 16, no. 12, pp. 11–14, Dec 1983.

[19] L. Chen and M. B. Tahoori, "An Efficient Probability Framework for Error Propagation and Correlation Estimation," in *International On-Line Testing Symposium*, pp. 70–75, 2012.

[20] L. Chen, M. Ebrahimi, and M. Tahoori, "CEP: Correlated Error Propagation for Hierarchical Soft Error Analysis," *Journal of Electronic Testing*, vol. 29, no. 2, pp. 143–158, 2013.

[21] L. Chen, F. Firouzi, S. Kiamehr, and M. Tahoori, "Fast and Accurate Soft Error Rate Estimation at RTL level," in *GMM/GI/ITG-Fachtagung Zuverlaessigkeit und Entwurf*, 2011.

[22] L. Chen, M. Ebrahimi, and M. Tahoori, "Quantitative Evaluation of Register Vulnerabilities in RTL Control Paths," in *European Test Symposium*, pp. 1–2, 2014.

[23] L. Chen, M. Ebrahimi, and M. Tahoori, "Formal Quantification of the Register Vulnerabilities to Soft Error in RTL Control Paths," *Journal of Electronic Testing*, vol. 31, no. 2, pp. 193–206, 2015.

[24] L. Chen and M. Tahoori, "Reliability-aware Register Binding for Control-Flow Intensive Designs," in *Design Automation Conference*, pp. 75:1–75:6, 2014.

[25] L. Chen, M. Ebrahimi, and M. Tahoori, "Reliability-aware Resource Allocation and Binding in High Level Synthesis," *ACM Transaction on Design Automation of Electronic Systems*, p. Submitted for review, 2015.

[26] S. Borkar, "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation," *IEEE Micro*, vol. 25, no. 6, pp. 10–16, 2005.

[27] S. Kumar, S. Agarwal, and J. P. Jung, "Soft error issue and importance of low alpha solders for microelectronics packaging," *Reviews on Advanced Materials Science*, vol. 34, pp. 184–202, 2013.

[28] J. Autran and D. Munteanu, *Soft Errors: From Particles to Circuits*. Taylor and Francis, 2015.

[29] "Measurement and reporting of alpha particle and terrestrial cosmic ray-induced soft errors in semiconductor devices," JEDEC Standard JESD89A, 2006.

[30] B. W. Johnson, *Design and Analysis of Fault Tolerant Digital Systems*. Addison-Wesley Longman Publishing Co., Inc., 1988.

[31] R. Baumann, "Landmarks in terrestrial single event effects," in *IEEE NSREC Short Course*, 2013.

[32] "International technology roadmap for semiconductors," http://www.itrs.net, 2012.

[33] S. Mitra, P. Bose, E. Cheng, C.-Y. Cher, H. Cho, R. Joshi, Y. Kim, C. Lefurgy, Y. Li, K. Rodbell, K. Skadron, J. Stathis, and L. Szafaryn, "The resilience wall: Cross-layer solution strategies," in *International Symposium on VLSI Technology, Systems and Application*, pp. 1–11, April 2014.

[34] L. Leem, H. Cho, H.-H. Lee, Y. M. Kim, Y. Li, and S. Mitra, "Cross-layer error resilience for robust systems," in *International Conference on Computer-Aided Design*, pp. 177–180, 2010.

[35] R. Harada, Y. Mitsuyama, M. Hashimoto, and T. Onoye, "Neutron induced single event multiple transients with voltage scaling and body biasing," in *IEEE International Reliability Physics Symposium*, pp. 3C.4.1–3C.4.5, 2011.

[36] N. Mahatme, S. Jagannathan, T. Loveless, L. Massengill, B. Bhuva, S.-J. Wen, and R. Wong, "Comparison of combinational and sequential error rates for a deep submicron process," *IEEE Transaction on Nuclear Science*, vol. 58, no. 6, pp. 2719–2725, dec. 2011.

[37] K. Lilja, M. Bounasser, S.-J. Wen, R. Wong, J. Holst, N. Gaspard, S. Jagannathan, D. Loveless, and B. Bhuva, "Single-event performance and layout optimization of flip-flops in a 28-nm bulk technology," *IEEE Transactions on Nuclear Science*, vol. 60, no. 4, pp. 2782–2788, Aug 2013.

[38] S. Mitra, N. Seifert, M. Zhang, Q. Shi, and K. S. Kim, "Robust System Design with Built-in Soft-error Resilience," *IEEE Computer*, vol. 38, pp. 43–52, 2005.

[39] D. D. Gajski, S. Abdi, A. Gerstlauer, and G. Schirner, *Embedded System Design: Modeling, Synthesis and Verification*. Springer, 2009.

[40] G. D. Micheli, *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, 1994.

[41] A. Dharchoudhury, S. Kang, H. Cha, and J. Patel, "Fast timing simulation of transient faults in digital circuits," in *International Conference on Computer-Aided Design*, pp. 719–726, Nov 1994.

[42] G. Messenger, "Collection of charge on junction nodes from ion tracks," *IEEE Transactions on Nuclear Science*, vol. 29, no. 6, pp. 2024–2031, 1982.

[43] P. Dodd and L. Massengill, "Basic mechanisms and modeling of single-event upset in digital microelectronics," *IEEE Transaction on Nuclear Science*, vol. 50, no. 3, pp. 583–602, 2003.

[44] K. Yamaguchi, Y. Takemura, K. Osada, K. Ishibashi, and Y. Saito, "3-d device modeling for sram soft-error immunity and tolerance analysis," *IEEE Transactions on Electron Devices*, vol. 51, no. 3, pp. 378–388, 2004.

[45] P. Shivakumar, M. Kistler, S. Keckler, D. Burger, and L. Alvisi, "Modeling the effect of technology trends on the soft error rate of combinational logic," in *International Conference on Dependable Systems and Networks*, pp. 389–398, 2002.

[46] R. Rao, K. Chopra, D. Blaauw, and D. Sylvester, "An efficient static algorithm for computing the soft error rates of combinational circuits," in *Proceeding of Design, Automation and Test in Europe*, pp. 1–6, 2006.

[47] B. Zhang, W. Wang, and M. Orshansky, "FASER: fast analysis of soft error susceptibility for cell-based designs," in *International Symposium on Quality Electronic Design*, pp. 755–760, 2006.

[48] F. Wang, Y. Xie, R. Rajaraman, and B. Vaidyanathan, "Soft error rate analysis for combinational logic using an accurate electrical masking model," in *International Conference on VLSI Design*, pp. 165–170, 2007.

[49] E. Costenaro, D. Alexandrescu, K. Belhaddad, and M. Nicolaidis, "A practical approach to single event transient analysis for highly complex design," *Journal of Electronic Testing*, vol. 29, no. 3, pp. 301–315, 2013.

[50] S. Krishnaswamy, G. Viamontes, I. Markov, and J. Hayes, "Probabilistic transfer matrices in symbolic reliability analysis of logic circuits," *ACM Transaction on Design Automation of Electronic Systems*, vol. 13, no. 1, pp. 8:1–8:35, 2008.

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

[52] M. Choudhury and K. Mohanram, "Reliability analysis of logic circuits," *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 3, pp. 392–405, 2009.

[53] M. Ebrahimi, A. Evans, M. B. Tahoori, R. Seyyedi, E. Costenaro, and D. Alexandrescu, "Comprehensive analysis of alpha and neutron particle-induced soft errors in an embedded processor at nanoscales," in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 30:1–30:6, 2014.

[54] J. Baraza, J. Gracia, S. Blanc, D. Gil, and P. Gil, "Enhancement of fault injection techniques based on the modification of VHDL code," *IEEE Transaction on Very Large Scale Integration Systems*, vol. 16, no. 6, pp. 693–706, 2008.

[55] N. Wang, J. Quek, T. Rafacz, and S. Patel, "Characterizing the effects of transient faults on a high-performance processor pipeline," in *International Conference on Dependable Systems and Networks*, pp. 61–70, 2004.

[56] L. Entrena, M. Garcia-Valderas, R. Fernandez-Cardenal, A. Lindoso, M. Portela, and C. Lopez-Ongil, "Soft error sensitivity evaluation of microprocessors by multilevel emulation-based fault injection," *IEEE Transaction on Computers*, vol. 61, no. 3, pp. 313–322, march 2012.

[57] N. Miskov-Zivanov and D. Marculescu, "Circuit reliability analysis using symbolic techniques," *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 12, pp. 2638–2649, 2006.

[58] M. Zhang and N. Shanbhag, "Soft-Error-Rate-Analysis (SERA) methodology," *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 10, pp. 2140–2155, 2006.

[59] R. Rajaraman, J. S. Kim, N. Vijaykrishnan, Y. Xie, and M. J. Irwin, "SEAT-LA: A soft error analysis tool for combinational logic," in *International Conference on VLSI Design*, pp. 499–502, 2006.

[60] X. Li, S. V. Adve, P. Bose, and J. A. Rivers, "SoftArch: an architecture-level tool for modeling and analyzing soft errors," in *International Conference on Dependable Systems and Networks*, pp. 496–505, 2005.

[61] T. Calin, M. Nicolaidis, and R. Velazco, "Upset hardened memory design for submicron cmos technology," *IEEE Transaction on Nuclear Science*, vol. 43, no. 6, pp. 2874–2878, 1996.

[62] 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," *IEEE Journal of Solid-State Circuits*, vol. 39, no. 9, pp. 1536–1543, 2004.

[63] N. Seifert, V. Ambrose, B. Gill, Q. Shi, R. Allmon, C. Recchia, S. Mukherjee, N. Nassif, J. Krause, J. Pickholtz, and A. Balasubramanian, "On the radiation-induced soft error performance of hardened sequential elements in advanced bulk cmos technologies," in *International Reliability Physics Symposium*, pp. 188–197, 2010.

[64] Y. Dhillon, A. Diril, A. Chatterjee, and A. Singh, "Analysis and optimization of nanometer cmos circuits for soft-error tolerance," *IEEE Transaction on Very Large Scale Integration Systems*, vol. 14, no. 5, pp. 514 –524, 2006.

[65] Q. Zhou and K. Mohanram, "Cost-effective radiation hardening technique for combinational logic," in *International Conference on Computer Aided Design*, pp. 100–106, Nov 2004.

[66] V. Sridharan and D. Kaeli, "Eliminating microarchitectural dependency from architectural vulnerability," in *International Symposium on High Performance Computer Architecture*, pp. 117–128, Feb 2009.

[67] S. Kim and A. Somani, "Soft Error Sensitivity Characterization for Microprocessor Dependability Enhancement Strategy," in *International Conference on Dependable Systems and Networks*, pp. 416–428, 2002.

[68] N. Wang, A. Mahesri, and S. Patel, "Examining ACE analysis reliability estimates using fault-injection," in *ACM SIGARCH Computer Architecture News*, vol. 35, no. 2, pp. 460–469, 2007.

[69] U. Krautz, M. Pflanz, C. Jacobi, H. W. Tast, K. Weber, and H. T. Vierhaus, "Evaluating coverage of error detection logic for soft errors using formal methods," in *Proceeding of Design, Automation and Test in Europe*, pp. 176–181, 2006.

[70] S. Mukherjee, C. Weaver, J. Emer, S. Reinhardt, and T. Austin, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," in *International Symposium on Microarchitecture*, pp. 29 – 40, dec. 2003.

[71] A. Haghdoost, H. Asadi, and A. Baniasadi, "System-level vulnerability estimation for data caches," in *Pacific Rim International Symposium on Dependable Computing*, pp. 157–164, 2010.

[72] L. Tang, S. Wang, J. Hu, and X. S. Hu, "Characterizing the L1 Data Cache's Vulnerability to Transient Errors in Chip-Multiprocessors," in *International Symposium on VLSI*, pp. 266–271, 2011.

[73] S. Wang, "Characterizing system-level vulnerability for instruction caches against soft errors," in *Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, pp. 356–363, 2011.

[74] Y. Cheng, M. Anguo, and M. Zhang, "Accurate and simplified prediction of l2 cache vulnerability for cost-efficient soft error protection," *IEICE transactions on Information and Systems*, vol. 95, no. 1, pp. 56–66, 2012.

[75] P. Montesinos, W. Liu, and J. Torrellas, "Using register lifetime predictions to protect register files against soft errors," in *International Conference on Dependable Systems and Networks*, pp. 286–296, 2007.

[76] N. J. George, C. R. Elks, B. W. Johnson, and J. Lach, "Transient fault models and avf estimation revisited," in *International Conference on Dependable Systems and Networks*, pp. 477–486, 2010.

[77] M. Ebrahimi, L. Chen, H. Asadi, and M. Tahoori, "CLASS: Combined Logic and Architectural Soft Error Sensitivity Analysis," in *Asia and South Pacific Design Automation Conference*, pp. 1–6, 2013.

[78] G. Fey, A. Sülflow, S. Frehse, and R. Drechsler, "Effective robustness analysis using bounded model checking techniques," *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 8, pp. 1239–1252, 2011.

[79] P. Meaney, S. Swaney, P. Sanda, and L. Spainhower, "IBM z990 soft error detection and recovery," *IEEE Transaction on Device and Materials Reliability*, vol. 5, no. 3, pp. 419–427, 2005.

[80] S. Baeg, S. Wen, and R. Wong, "SRAM interleaving distance selection with a soft error failure model," *IEEE Transaction on Nuclear Science*, vol. 56, no. 4, pp. 2111–2118, 2009.

[81] S. Tosun, N. Mansouri, E. Arvas, M. Kandemir, and Y. Xie, "Reliability-Centric High-Level Synthesis," in *Proceeding of Design, Automation and Test in Europe*, pp. 1258–1263, 2005.

[82] M. Glaß, M. Lukasiewycz, T. Streichert, C. Haubelt, and J. Teich, "Reliability-aware System Synthesis," in *Proceeding of Design, Automation and Test in Europe*, pp. 409–414, 2007.

[83] S. Golshan, H. Kooti, and E. Bozorgzadeh, "SEU-Aware High-Level Data Path Synthesis and Layout Generation on SRAM-Based FPGAs," *IEEE Transaction on Computer-Aided Design*, vol. 30, no. 6, pp. 829–840, 2011.

[84] T. Imagawa, H. Tsutsui, H. Ochi, and T. Sato, "A Cost-effective Selective TMR for Heterogeneous Coarse-grained Reconfigurable Architectures Based on DFG-level Vulnerability Analysis," in *Design, Automation and Test in Europe*, pp. 701–706, 2013.

[85] G. Lakshminarayana, A. Raghunathan, and N. K. Jha, "Behavioral Synthesis of Fault Secure Controller Datapaths Based on Aliasing Probability Analysis," *IEEE Transaction on Computer*, vol. 49, no. 9, pp. 865–885, 2000.

[86] K. Wu and R. Karri, "Fault Secure Datapath Synthesis Using Hybrid Time and Hardware Redundancy," *IEEE Transaction on Computer-Aided Design*, vol. 23, no. 10, pp. 1476–1485, 2006.

[87] G. Reis, J. Chang, N. Vachharajani, R. Rangan, and D. August, "Swift: software implemented fault tolerance," in *International Symposium on Code Generation and Optimization*, pp. 243–254, March 2005.

[88] O. Goloubeva, *Software-Implemented Hardware Fault Tolerance*. Springer, 2006.

[89] M. Rebaudengo, M. Reorda, M. Torchiano, and M. Violante, "Soft-error detection through software fault-tolerance techniques," in *International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp. 210–218, Nov 1999.

[90] M. Hiller, "Executable assertions for detecting data errors in embedded control systems," in *Proceedings International Conference on Dependable Systems and Networks*, pp. 24–33, 2000.

[91] Z. Alkhalifa, V. Nair, N. Krishnamurthy, and J. Abraham, "Design and evaluation of system-level checks for on-line control flow error detection," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 6, pp. 627–641, Jun 1999.

[92] N. Oh, P. Shirvani, and E. McCluskey, "Control-flow checking by software signatures," *IEEE Transactions on Reliability*, vol. 51, no. 1, pp. 111–122, Mar 2002.

[93] U. Kretzschmar, A. Astarloa, J. Lazaro, J. Jimenez, and A. Zuloaga, "An automatic experimental set-up for robustness analysis of designs implemented on SRAM FPGAs," in *International Symposium on System on Chip*, pp. 96–101, 2011.

[94] L. Chen and M. B. Tahoori, "Soft Error Propagation and Correlation Estimation in Combinational Network," in *Workshop on Manufacturable and Dependable Multicore Architectures at Nanoscale*, 2012.

[95] H. Asadi and M. Tahoori, "Soft Error Derating Computation in Sequential Circuits," in *International Conference on Computer-Aided Design*, pp. 497–501, 2006.

[96] T. Rejimon, K. Lingasubramanian, and S. Bhanja, "Probabilistic error modeling for nano-domain logic circuits," *IEEE Transaction on Very Large Scale Integration Systems*, vol. 17, no. 1, pp. 55–65, 2009.

[97] S. Sivaswamy, K. Bazargan, and M. Riedel, "Estimation and optimization of reliability of noisy digital circuits," in *International Symposium on Quality of Electronic Design*, pp. 213–219, 2009.

[98] C. Yu and J. Hayes, "Trigonometric method to handle realistic error probabilities in logic circuits," in *Proceeding of Design, Automation and Test in Europe*, pp. 1–6, 2011.

[99] N. Mohyuddin, E. Pakbaznia, and M. Pedram, "Probabilistic error propagation in logic circuits using the Boolean difference calculus," in *IEEE International Conference on Computer Design*, pp. 7–13, 2008.

[100] R. Marculescu, D. Marculescu, and M. Pedram, "Probabilistic modeling of dependencies during switching activity analysis," *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 2, pp. 73–83, 1998.

[101] A. Papoulis, *Probability and statistics*. Prentice Hall, 1990.

[102] S. Ercolani, M. Favalli, M. Damiani, P. Olivo, and B. Ricco, "Estimate of signal probability in combinational logic networks," in *European Test Conference*, pp. 132–138, 1989.

[103] J. C. Costa, L. M. Silveira, S. Devadas, and J. Monteiro, "Power estimation using probability polynomials," *Design Automation for Embedded Systems*, pp. 19–52, 2004.

[104] N. Miskov-Zivanov and D. Marculescu, "Multiple transient faults in combinational and sequential circuits: a systematic approach," *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 10, pp. 1614–1627, 2010.

[105] M. Fazeli, S. Ahmadian, S. Miremadi, H. Asadi, and M. Tahoori, "Soft error rate estimation of digital circuits in the presence of Multiple Event Transients (METs)," in *Design, Automation and Test in Europe Conference*, pp. 1–6, march 2011.

[106] J. Han, H. Chen, E. Boykin, and J. Fortes, "Reliability evaluation of logic circuits using probabilistic gate models," *Microelectronics Reliability*, vol. 51, no. 2, pp. 468 – 476, 2011.

[107] G. Csárdi and T. Nepusz, "The igraph software package for complex network research," *InterJournal Complex Systems*, p. 1695, 2006.

[108] R. Rubinstein and D. Kroese, *Simulation and the Monte Carlo method*. John Wiley, 2008.

[109] OpenRISC, http://opencores.org/openrisc.

[110] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *IEEE International Workshop on Workload Characterization*, pp. 3–14, 2001.

[111] S. Bhanja and N. Ranganathan, "Cascaded bayesian inferencing for switching activity estimation with correlated inputs," *IEEE Transaction on Very Large Scale Integration Systems*, vol. 12, no. 12, pp. 1360–1370, 2004.

[112] D. Holcomb, W. Li, and S. A. Seshia, "Design as you See FIT: System-level soft error analysis of sequential circuits," in *International Conference on Design, Automation and Test in Europe*, pp. 785–790, 2009.

[113] N. Miskov-Zivanov and D. Marculescu, "Modeling and optimization for soft-error reliability of sequential circuits," *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 5, pp. 803–816, 2008.

[114] I. Polian, J. Hayes, S. Reddy, and B. Becker, "Modeling and mitigating transient errors in logic circuits," *IEEE Transaction on Dependable and Secure Computing*, vol. 8, no. 4, pp. 537–547, 2011.

[115] J. Liang, J. Han, and F. Lombardi, "Analysis of error masking and restoring properties of sequential circuits," *IEEE Transaction on Computers*, vol. 62, no. 9, pp. 1694–1704, 2013.

[116] G. Lakshminarayana, A. Raghunathan, N. K. Jha, and S. Dey, "Transforming control-flow intensive designs to facilitate power management," in *International Conference on Computer-aided Design*, pp. 657–664, 1998.

[117] Y. Li, E. Cheng, S. Makar, and S. Mitra, "Self-repair of uncore components in robust system-on-chips: An opensparc t2 case study," in *International Test Conference*, pp. 1–10, 2013.

[118] A. Evans, D. Alexandrescu, E. Costenaro, and L. Chen, "Hierarchical RTL-based combinatorial SER estimation," in *International On-Line Testing Symposium*, pp. 139–144, 2013.

[119] L. Anghel, R. Leveugle, and P. Vanhauwaert, "Evaluation of SET and SEU effects at multiple abstraction levels," in *International On-Line Testing Symposium*, pp. 309–312, 2005.

[120] S. Bergaoui, P. Vanhauwaert, and R. Leveugle, "A New Critical Variable Analysis in Processor-Based Systems," *IEEE Transaction on Nuclear Science*, vol. 57, pp. 1992–1999, 2010.

[121] J. Cong and K. Gururaj, "Assuring application-level correctness against soft errors," in *International Conference on Computer-Aided Design*, pp. 150–157, 2010.

[122] J. Lee and A. Shrivastava, "Static Analysis of Register File Vulnerability," *IEEE Transaction Computer Aided Design Integrated Circuits and Systems*, vol. 30, no. 4, pp. 607–616, 2011.

[123] M. Shafique, S. Rehman, P. V. Aceituno, and J. Henkel, "Exploiting Program-level Masking and Error Propagation for Constrained Reliability Optimization," in *Design Automation Conference*, pp. 17:1–17:9, 2013.

[124] S. Rehman, M. Shafique, F. Kriebel, and J. Henkel, "Reliable software for unreliable hardware: Embedded code generation aiming at reliability," in *International Conference on Hardware/Software Codesign and System Synthesis*, pp. 237–246, 2011.

[125] P. Chu, *RTL Hardware Design Using VHDL: Coding for Efficiency, Portability, and Scalability.* Wiley, 2006.

[126] H. Lin, C. Wang, S. Chang, Y. Chen, H. Chou, C. Huang, Y. Yang, and C. Shen, "A Probabilistic Analysis Method for Functional Qualification under Mutation Analysis," in *Design, Automation Test in Europe Conference*, pp. 147 –152, march 2012.

[127] C. Baier and J. Katoen, *Principles of Model Checking.* MIT Press, 2008.

[128] J. A. Kumar and S. Vasudevan, "Automatic compositional reasoning for probabilistic model checking of hardware designs," in *International Conference on the Quantitative Evaluation of Systems*, pp. 143–152, 2010.

[129] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of Probabilistic Real-time Systems," in *International Conference on Computer Aided Verification*, 2011.

[130] R. Wimmer, M. Herbstritt, H. Hermanns, K. Strampp, and B. Becker, "Sigref – a symbolic bisimulation tool box," in *International Symposium on Automated Technology for Verification and Analysis*, pp. 477–492, 2006.

[131] A. Aziz, V. Singhal, and F. Balarin, "It usually works: The temporal logic of stochastic systems," in *International Conference on Computer Aided Verification*, pp. 155–165, 1995.

[132] J. Katoen, T. Kemna, I. Zapreev, and D. Jansen, "Bisimulation minimisation mostly speeds up probabilistic model checking," in *International Conference on tools and algorithms for the construction and analysis of systems*, pp. 87–101, 2007.

[133] C. Dehnert, J. Katoen, and D. Parker, "SMT-based bisimulation minimisation of markov models," in *VMCAI*, pp. 28–47, 2013.

[134] K. L. McMillan, "A methodology for hardware verification using compositional model checking," *Journal Science of Computer Programming*, vol. 37, no. 1-3, pp. 279–309, 2000.

[135] P. Bjesse, "Word level bitwidth reduction for unbounded hardware model checking," *Formal Methods System Design*, vol. 35, no. 1, pp. 56–72, Aug 2009.

[136] A. Mishchenko, M. Case, R. Brayton, and S. Jang, "Scalable and scalably-verifiable sequential synthesis," in *International Conference on Computer-Aided Design*, pp. 234–241, 2008.

[137] G. Karypis and V. Kumar, "Multilevel k-way hypergraph partitioning," in *Design Automation Conference*, pp. 343–348, 1999.

[138] J. Kumar and S. Vasudevan, "Verifying dynamic power management schemes using statistical model checking," in *Asia and South Pacific Design Automation Conference*, pp. 579–584, 2012.

[139] G. Asadi and M. B. Tahoori, "An analytical approach for soft error rate estimation in digital circuits," in *Proceeding of IEEE International Symposium Circuits and Systems*, pp. 2991–2994, 2005.

[140] L.-T. Wang, C.-W. Wu, and X. Wen, *VLSI Test Principles and Architectures: Design for Testability*. Morgan Kaufmann Publishers Inc., 2006.

[141] R. C. Cheung, "A User-Oriented Software Reliability Model," *IEEE Transaction on Software Engineering*, vol. 6, no. 2, pp. 118–125, 1980.

[142] A. Filieri, C. Ghezzi, V. Grassi, and R. Mirandola, "Reliability Analysis of Component-based Systems with Multiple Failure Modes," in *Conference Component-Based Software Engineering*, pp. 1–20, 2010.

[143] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools*, 2nd ed. Addison-Wesley, 2007.

[144] OpenSPARC, http://www.oracle.com/technetwork/systems/opensparc.

[145] N. T. Clark, H. Zhong, and S. A. Mahlke, "Automated custom instruction generation for domain-specific processor acceleration," *IEEE Transaction on Computers*, vol. 54, no. 10, pp. 1258–1270, 2005.

[146] H. Noori, F. Mehdipour, M. S. Zamani, K. Inoue, and K. Murakami, "Handling control data flow graphs for a tightly coupled reconfigurable accelerator," in *Proceeding of the 3rd International Conference on Embedded Software and Systems*, pp. 249–260, 2007.

[147] V. Sundararajan and K. K. Parhi, "Synthesis of low power folded programmable coefficient fir digital filters (short paper)," in *Proceeding of the Asia and South Pacific Design Automation Conference*, pp. 153–156, 2000.

[148] M. Fazeli, S. Miremadi, H. Asadi, and S. Ahmadian, "A fast and accurate multi-cycle soft error rate estimation approach to resilient embedded systems design," in *International Conference on Dependable Systems and Networks*, pp. 131–140, 2010.

[149] D. Bhaduri, S. Shukla, P. Graham, and M. Gokhale, "Reliability analysis of large circuits using scalable techniques and tools," *IEEE Transaction on Circuits and Systems*, vol. 54, no. 11, pp. 2447–2460, 2007.

[150] Z. Dan, "Research on sequential equivalence checking based system-level soft error reliability analysis of circuits," Ph.D. dissertation, National University of Defense Technology, Hunan, China, 2010.

[151] J. Hayes, I. Polian, and B. Becker, "An analysis framework for transient-error tolerance," in *VLSI Test Symposium*, pp. 249–255, 2007.

[152] J. Luo, L. Zhong, Y. Fei, and N. Jha, "Register binding-based RTL power management for control-flow intensive designs," *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 8, pp. 1175–1183, 2004.

[153] A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, T. Czajkowski, S. D. Brown, and J. H. Anderson, "Legup: An open-source high-level synthesis tool for fpga-based processor/accelerator systems," *ACM Transaction in Embedded Computing Systems*, vol. 13, no. 2, pp. 24:1–24:27, 2013.

[154] F. F. Hsu, E. M. Rudnick, and J. H. Patel, "Enhancing High-level Control-flow for Improved Testability," in *International Conference on Computer-Aided Design*, pp. 322–328, 1996.

[155] F. Gao and J. P. Hayes, "Exact and Heuristic Approaches to Input Vector Control for Leakage Power Reduction," *IEEE Transaction on Computer-Aided Design*, vol. 25, no. 11, pp. 2564–2571, 2006.

[156] M. Rim, A. Mujumdar, R. Jain, and R. De Leone, "Optimal and Heuristic Algorithms for Solving the Binding Problem," *IEEE Transaction on Very Large Scale Integrated Systems*, vol. 2, no. 2, pp. 211–225, 1994.

[157] L. Kruse, E. Schmidt, G. Jochens, A. Stammermann, A. Schulz, E. Macii, and W. Nebel, "Estimation of lower and upper bounds on the power consumption from scheduled data flow graphs," *IEEE Transaction on Very Large Scale Integrated Systems*, vol. 9, no. 1, pp. 3–15, 2001.

[158] Y. Hara, H. Tomiyama, S. Honda, and H. Takada, "Proposal and quantitative analysis of the chstone benchmark program suite for practical c-based high-level synthesis," *Journal of Information Processing*, vol. 17, pp. 242–254, 2009.

[159] C. Lattner and V. Adve, "LLVM: A Compilation Framework for Lifelong Program Analysis and Transformation," in *International Symposium on Code Generation and Optimization*, 2004.

[160] TFIT, http://www.iroctech.com/soft-error-tools/tfit-cell-level/, 2014.

[161] N. Mahatme, N. Gaspard, S. Jagannathan, T. Loveless, B. Bhuva, W. Robinson, L. Massengill, S.-J. Wen, and R. Wong, "Impact of supply voltage and frequency on the soft error rate of logic circuits," *IEEE Transactions on Nuclear Science*, vol. 60, no. 6, pp. 4200–4206, Dec 2013.

[162] J. Cong and Z. Zhang, "An Efficient and Versatile Scheduling Algorithm Based on SDC Formulation," in *Design Automation Conference*, pp. 433–438, 2006.

[163] Y. Hara-Azumi and H. Tomiyama, "Cost-efficient scheduling in high-level synthesis for Soft-Error Vulnerability Mitigation," in *International Symposium on Quality Electronic Design*, pp. 502–507, 2013.

[164] S. Tosun, O. Ozturk, N. Mansouri, E. Arvas, M. Kandemir, Y. Xie, and W.-L. Hung, "An ilp formulation for reliability-oriented high-level synthesis," in *International Symposium on Quality of Electronic Design*, pp. 364–369, 2005.

[165] G. Saggese, A. Vetteth, Z. Kalbarczyk, and R. Iyer, "Microprocessor sensitivity to failures: control vs. execution and combinational vs. sequential logic," in *International Conference on Dependable Systems and Networks*, pp. 760–769, June 2005.

[166] N. Nakka, K. Pattabiraman, and R. Iyer, "Processor-level selective replication," in *International Conference on Dependable Systems and Networks*, pp. 544–553, 2007.

[167] S. Gupta, R. Gupta, N. Dutt, and A. Nicolau, *SPARK: a parallelizing approach to the high-level synthesis of digital circuits*. Springer, 2004.

[168] C.-T. Hwang, J.-H. Lee, and Y.-C. Hsu, "A formal approach to the scheduling problem in high level synthesis," *IEEE Transaction on Computer-Aided Design*, vol. 10, no. 4, pp. 464–475, 1991.

[169] F. Khosravi, F. Reimann, M. Glass, and J. Teich, "Multi-objective local-search optimization using reliability importance measuring," in *Design Automation Conference*, pp. 1–6, 2014.

[170] lp_solve, http://sourceforge.net/projects/lpsolve.

[171] A. Dixit and A. Wood, "The impact of new technology on soft error rates," in *International Reliability Physics Symposium*, pp. 4.1–4.7, 2011.