University of California

Los Angeles

Logic Synthesis for FPGA Reliability

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Electrical Engineering

by

Zhe Feng

2013

Abstract of the Dissertation

Logic Synthesis for FPGA Reliability

by

Zhe Feng

Doctor of Philosophy in Electrical Engineering

University of California, Los Angeles, 2013

Professor Lei He, Chair

Logic synthesis is one of the key stages in the computer-aided design (CAD) flow for a field programmable gate array (FPGA) based design. It usually consists of a series of optimization iterations to improve the quality of results (QoR) of the design. Besides the traditional optimization objectives (e.g., performance, area, power), the reliability is becoming a main concern as modern FPGAs have advanced to 20nm technology, due to reduction in core voltage, decrease in transistor geometry, and increase in switching speed. However, existing techniques for enhancing the reliability of FPGA based designs fall behind industrial needs in terms of cost (e.g., area and power overhead), CAD flow, runtime, and the FPGA architecture.

To address the problems, this dissertation proposes several novel logic synthesis algorithms. The first algorithm seeks a formal method to improve the reliability of FPGA based designs while incurring minimal area and power overhead. The algorithm formulates the problem of the FPGA reliability under random faults as a stochastic satisfiability (SSAT) based Boolean matching, and employs robust templates to rewrite the look-up table (LUT) based netlist, to maximize the stochastic yield rate. To ensure not breaking the current CAD flow, a logic synthesis algorithm is presented that performs a SAT-based in-place reconfiguration in the LUT to mask soft errors, without changing of the functionality and topology of the LUT based netlist. In addition, the dissertation proposes three fast in-place logic synthesis algorithms targeting the modern FPGA architecture including both LUTs and interconnects, which perform simulation guided netlist analyses and utilize don't cares in the netlist to enhance the reliability of the design. The effectiveness of the proposed algorithms are verified by experimental results.

The dissertation of Zhe Feng is approved.

Milos D. Ercegovac

Puneet Gupta

Kung Yao

Lei He, Committee Chair

University of California, Los Angeles

2013

To My Family and Friends.

Table of Contents

List of Figures

List of Tables

Acknowledgments

During the development of the dissertation, I have received support and encouragement from a great number of individuals. I'd like to acknowledge them here, with my full gratitude.

I would like to express the deepest appreciation to my advisor, my committee chair, Professor Lei He, for his generous help over the years. His continuous guidance and encouragement foster an environment of academic excellence that helps me develop the problem analysing and solving skills, and eventually grow into an independent researcher. This dissertation should not have come to its completion without his tremendous support.

I would like to thank my doctoral committee members, Professor Kung Yao, Professor Puneet Gupta, and Professor Milos Ercegovac for their suggestions to improve this work. Their time and effort are greatly appreciated.

I am indebted to Professor Rupak Majumdar for his feedback as a formal verification expert. My special thanks to Dr. Paul Fang for his mentoring during my internship at Synopsys, Inc. Professor Jason Cong and Professor Zhiru Zhang earn my thanks as my mentors and collaborators when I was an intern at AutoESL Design Technology, Inc. My thanks to Dr. Babette van Antwerpen for her guidance during my stay in Altera, Inc.

My sincere thanks to Dr. Yu Hu, who helped on the part of algorithm development in Chapter 3, Dr. Naifeng Jing, who helped on the part of the experiments in Chapter 5.

I treasure the opportunity that I have worked with a group of wonderful and brilliant people at the UCLA Design Automation Lab. In particular, I thank Dr. Yan Lin, Dr. Lerong Cheng, Dr. Yiyu Shi, Dr. Wei Yao, Mr. Ju-Yueh Lee, Dr. Fang Gong, Mr. Wenyao Xu, Dr. Victor Shih, Mr. Zhen Cao for being around with helpful hands.

Last, and most of all, I am grateful to my family. I thank my wife Han Wang in particular. This dissertation becomes so humble compared to her continuous support and love.

Vita

2000-2004	B.S., Department of Computer Science, Northeastern University, Shenyang, China.
2004-2007	M.S., Department of Computer Science, Tsinghua University, Beijing, China.
2007–present	Teaching Assistant, Graduate Student Researcher, Department of Electrical Engineering, University of California, Los Angeles, California, USA.
2008	Intern, Synopsys, Inc., 700 E Middlefield Rd, Mountain View, CA, USA
2010	Intern, AutoESL Design Technology, Inc., 11835 W Olympic Blvd, Los Angeles, CA, USA
2011	Intern, Altera, Inc., 101 Innovation Dr, San Jose, CA, USA

Publications

Zhe Feng, Naifeng Jing, and Lei He, "IPF: In-Place X-Filling Algorithm for the Reliability of Modern FPGAs", *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, under second review, 2013.

Naifeng Jing, Ju-Yueh Lee, Zhe Feng, Weifeng He, Zhigang Mao, and Lei He, "SEU Fault Evaluation and Characteristics for SRAM-based FPGA Architectures and Synthesis Algorithms", *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2012.

Zhen Cao, Tom Tong Jing, Jinjun Xiong, Yu Hu, Zhe Feng, Lei He, and Xianlong Hong, "Fashion: A Fast and Accurate Solution to Global Routing Problem", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2008.

Zhe Feng, Naifeng Jing, Yu Hu, and Lei He, "IPF: In-place X-Filing to Mitigate Soft Errors in SRAM-based FPGAs", *International Conference on Field Programmable Logic and Applications (FPL)*, 2011.

Naifeng Jing, Ju-Yueh Lee, Zhe Feng, Weifeng He, Zhigang Mao, Shi-Jie Wen, Rick Won, and Lei He, "Quantitative SEU Fault Evaluation for SRAM-Based FPGA Architectures and Synthesis Algorithms", *International Conference on Field Programmable Logic and Applications (FPL)*, 2011.

Lintao Cui, Jing Chen, Yu Hu, Jinjun Xiong, Zhe Feng, and Lei He, "Acceleration of Multi-agent Simulation on FPGAs", *International Conference on Field Programmable Logic and Applications (FPL)*, 2011.

Ju-Yueh Lee, Zhe Feng, and Lei He, "In-Place Decomposition for Robustness in FPGA", *International Conference on Computer-Aided Design (ICCAD)*, 2010.

Zhe Feng, Yu Hu, Lei He, and Rupak Majumdar, "IPR: In-Place Reconfiguration for FPGA Fault Tolerance", *International Conference on Computer-Aided Design (ICCAD)*, 2009. (Best paper award nomination)

Yu Hu, Zhe Feng, Lei He, and Rupak Majumdar, "Robust FPGA Resynthesis Based on Fault Tolerant Boolean Matching", *International Conference on Computer-Aided Design (ICCAD)*, 2008. (Best paper award nomination)

Zhe Feng, Yu Hu, Rupak Majumdar, and Lei He, "IPR: In-Place Reconfiguration for FPGA Fault Tolerance", *IEEE International Workshop on Logic and Synthesis (IWLS)*, 2009.

Yu Hu, Zhe Feng, Rupak Majumdar, and Lei He, "Templates and Algorithms of Boolean Matching for Fault Tolerance in FPGAs", *IEEE International Workshop on Logic and Synthesis (IWLS)*, 2008.

CHAPTER 1

Introduction

1.1 Motivation

Due to fast time-to-market, no non-recurring engineering expense (NRE), and easy long-term maintenance, field programmable gate arrays (FPGAs) have been increasingly used in many applications, e.g., networking equipment, automotive systems, medical equipment, aerospace and defense applications. Besides, the ever-increasing integrated IP modules enable FPGAs surpass application specific integrated circuits (ASICs) in some aspects in terms of cost and performance. BDTI, a noted analyst and benchmarking firm, released benchmarks showing that Xilinx's FPGA is up to 34× more cost-effective than TI's DSP in a significant class of DSP applications [Ber06].

Traditionally, the optimization objectives for FPGA based designs are focused on performance, area, power, etc. Over the past decade, the reliability has been of increased interest in both academia and industry. Originally, it received attention only from mission critical applications in aerospace and defense realms. As modern FPGAs have advanced to 20nm technology to provide low cost and low power solutions [XIL13, Meh12, ALT12], they are prone to errors for most applications, because of reduction in core voltage, decrease in transistor geometry, and increase in switching speed [XIL12a]. Figure 1.1 [Bau05] shows the soft errors trend along the SRAM scaling. Although the number of errors on a single SRAM bit is deceasing, the system-level errors increase exponentially with respect to the SRAM

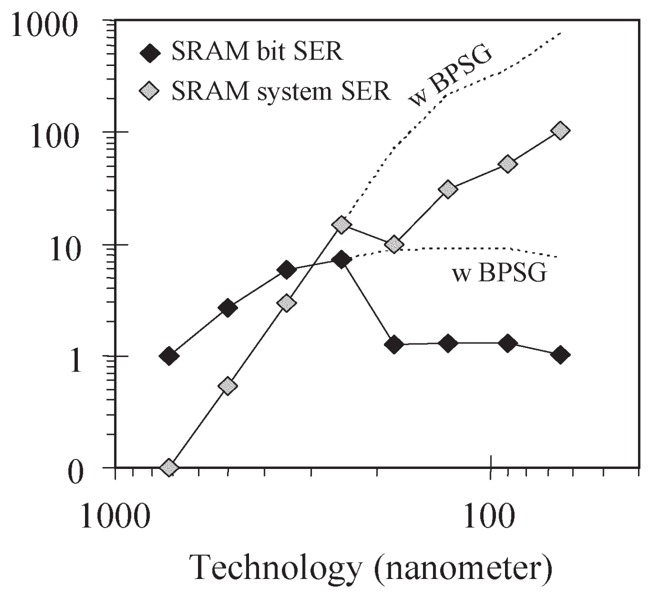


Figure 1.1: Soft errors vs. SRAM scaling trend.

technology. This is because the chip density (bits/system) increases at a faster speed than the error reduction on a single bit by technology scaling due to higher system requirements.

The errors can be categorized into hard errors and soft errors. Hard errors are permanent errors arising from circuit processing. Soft errors, also called single event upsets (SEUs), are generally caused by high-energy particle strikes, e.g., neutrons coming from cosmic rays or alpha particles emitting from trace impurities in packaging materials and solder bumps [Bid10]. Other reasons causing SEUs include the power supply disturbance and the electromagnetic interference. As shown in Figure 1.2 [FML04], SEUs change values of devices, such as SRAM cells and flip-flops, when charges collected from strikes are larger than the stored charges in the junctions for a long enough period of time. SEUs not only can change the values of SRAM cells storing user data, but it also can change the values of SRAM cells in control logic. Because most logic functions and interconnects in SRAM-based FPGAs are implemented by SRAM cells, they are more vulnerable to SEUs compared with ASICs. It implies that the reliability of FPGA based

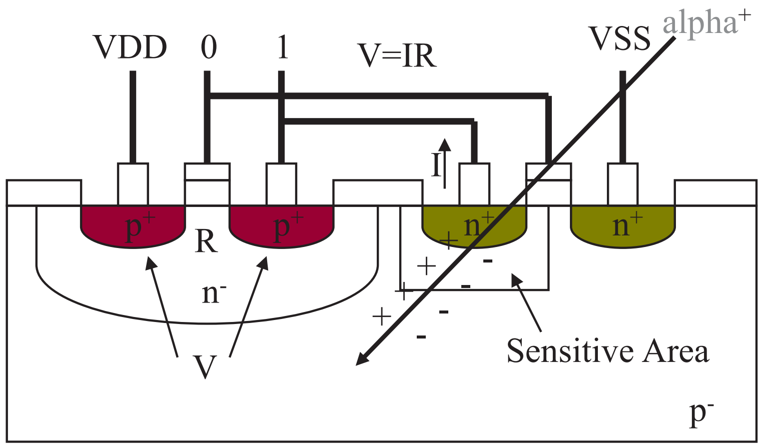


Figure 1.2: Alpha particle is striking on a SRAM cell and flipping the bit.

design cannot be sacrificed for the performance and cost of the system, and it must be accounted for as one of the optimization objectives during the design flow.

1.2 Dissertation Contributions

There are a number of techniques in the literature (listed in Section 2.3) for enhancing the reliability of FPGA based designs. However, most existing techniques fall behind industrial needs in terms of cost (e.g., area and power overhead), CAD flow, runtime, and FPGA architecture.

To address the problems, this dissertation proposes several novel logic synthesis algorithms taking the reliability as optimization objective, targeting both hard errors and soft errors. Specifically, we have made the following contributions,

- Targeting stochastic error rate reduction in the presence of both hard and soft errors while incurring minimal area and power overhead, we develop a formal method (i.e., stochastic satisfiability (SSAT)) based fault tolerant Boolean matching (FTBM), which exploits the flexibility of the look-up table (LUT) configuration in FPGAs to maximize the stochastic yield rate for a logic function. Using FTBM, we propose a robust resynthesis algorithm (ROSE) which maximizes stochastic yield rate for an entire circuit. Finally, we show that existing programmable logic block (PLB) templates for area-aware Boolean matching and logic resynthesis are not effective for fault tolerance, and we propose a new robust template with path re-convergence. Compared to the state-of-the-art academic technology mapper Berkeley ABC, ROSE using the proposed robust PLB template reduces the fault rate by 25% with 1% fewer LUTs, and increases mean time between failures (MTBF) by 31%, while preserving the optimal logic depth.

- To ensure not breaking the current CAD flow during the reliability optimization, we present a logic synthesis algorithm that performs a satisfiability (SAT) based in-place reconfiguration (IPR) in the LUT on FPGAs. IPR maximizes identical configuration bits for complementary inputs of a LUT to prevent the error propagation. It preserves the functionality and topology of the netlist, and therefore, requires no change to the physical design. Compared to the state-of-the-art academic technology mapper Berkeley ABC, IPR reduces the relative fault rate by 48% and increases MTTF by 1.94$\times$ with the same area and performance. Applying both ROSE and IPR reduces the relative fault rate by 49% and increases MTTF by 2.40$\times$ with 19% less area but same performance.

- Targeting the modern FPGA architecture including both LUTs and interconnects, we propose three synthesis based in-place X-Filling algorithms by utilizing don't cares to augment the reliability of FPGA based designs. Compared with circuit and architecture based solutions, our algorithms are in place, and do not require area, power, performance, and design time overheads. Compared with other synthesis based algorithms, we take into account the widely accepted interconnect architecture besides considering

LUTs during optimization. For the ten largest combinational MCNC benchmark circuits mapped to 6-LUTs, our approaches achieve up to a 37% greater failure rate reduction, and up to 150× runtime speedup, compared with the best known in-place algorithm, the In-Place Decomposition (IPD) algorithm.

1.3 Dissertation Outline

The rest of the dissertation is organized as follows.

Chapter 2 presents the technology background, including FPGA architecture, CAD flow, and a review of techniques for the FPGA reliability.

Chapter 3 presents an algorithm seeking a formal method to improve the reliability of FPGA based designs while incurring minimal area and power overhead. The algorithm formulates the problem of the reliability under random faults as a stochastic satisfiability (SSAT) based Boolean matching, and employs robust templates to rewrite the netlist, to maximize the stochastic yield rate.

In Chapter 4, to ensure not breaking the current CAD flow, a logic synthesis algorithm is presented which performs a SAT-based in-place reconfiguration in the LUT to mask soft errors, without changing the functionality and topology of the netlist.

Chapter 5 presents three fast in-place logic synthesis algorithms targeting the modern FPGA architecture including both LUTs and interconnects. The three algorithms perform simulation guided netlist analyses and utilize don't cares in the netlist to enhance the reliability of the design.

Chapter 6 concludes the dissertation with discussion of the ongoing work and future work as well.

CHAPTER 2

Technology Background

2.1 FPGA Architecture

A generic island-style FPGA architecture is shown in Figure 2.1. It consists of a 2D array of configurable logic blocks (CLBs) that are connected by programmable global routing architecture. Each CLB can be parameterized by (k, N), i.e., it consists of N LUTs and each LUT has k inputs. The configuration bits are in each LUT to implement the desired functionality. The LUT inputs and outputs are fully connected by intra-CLB routing (local routing) MUXes, which allow signals to be respectively routed from and to CLB inputs and outputs internally. The CLBs are connected via inter-CLB routing elements, i.e., switch boxes and connection boxes by wires deployed in routing channels with a width of w (the number of tracks). Wires in switch boxes and connections boxes together with wires in local routing, make up the whole interconnect architecture. Wires are linked by bidirectional and unidirectional programmable interconnect points (PIPs). Typically, the bidirectional PIPs are implemented by pass transistors, while unidirectional PIPs are the selection bits in MUXes. These configuration bits configuring PIPs contribute to most of the configuration bits in FPGA. Therefore, interconnects are critical to FPGA designs since routing structure contributes a large portion of the total FPGA area and configuration bits. There exits a large body of study exploring various FPGA architectures for different optimization objectives [KTR08]. In this dissertation, the most popular FPGA architecture [Meh12, ALT12] in modern industrial FPGAs, is assumed.

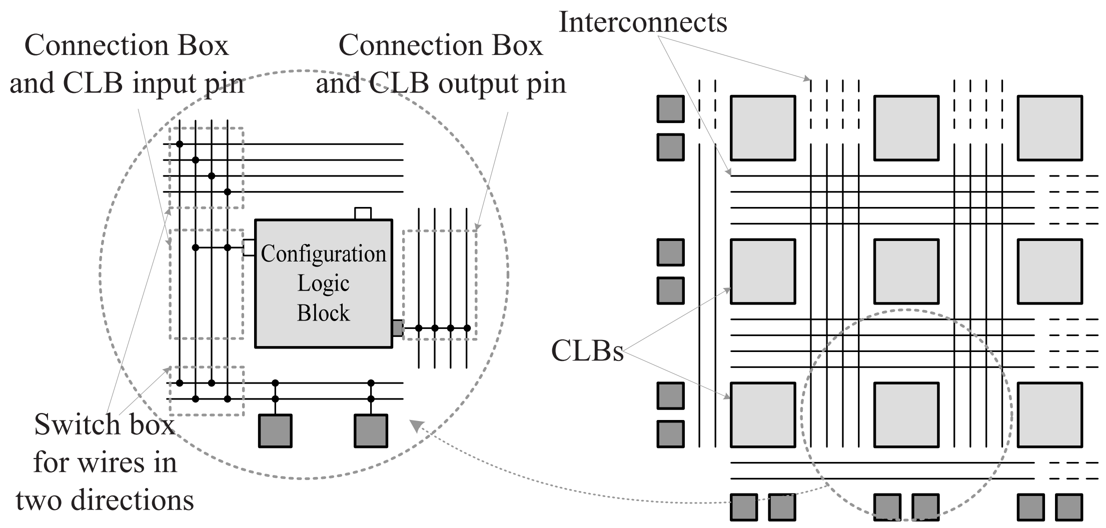


Figure 2.1: Island-style FPGA architecture

2.2 CAD Flow

CAD is one of the key factors to the QoR (e.g., performance, area, power, and reliability) of FPGA based designs. The synthesis procedures in the basic CAD flow for FPGAs are shown in Figure 2.2. They start with a description of the circuit, usually in the form of a hardware description language (HDL) such as VHDL and Verilog. Once the description is compiled, logic synthesis occurs where the description is synthesized into a gate-level netlist and then mapped to a netlist of programmable logic blocks (PLBs). After that, physical design, including placement and routing, is performed to arrange the PLB netlist into an FPGA. Sometimes, a physical synthesis may be performed between logic synthesis and physical design, which performs further optimization based on back annotated physical information from placement and routing.

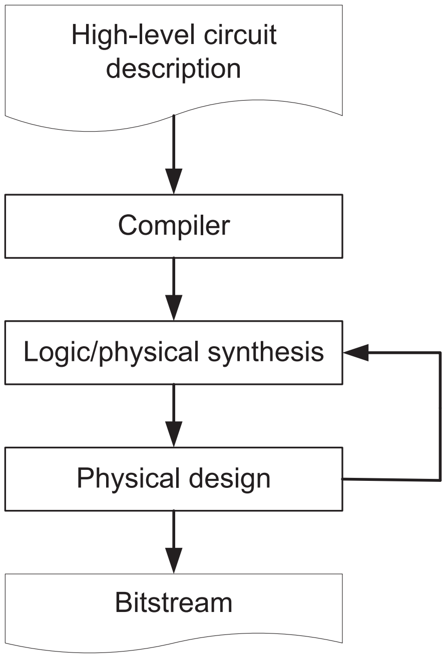


Figure 2.2: Generic CAD flow for FPGA based designs

2.3 Review of Techniques for FPGA Reliability

In the literature, there are extensive studies on the FPGA reliability [DH06]. They can be broadly categorized into two kinds, studies on error detection, evaluation, and prediction [RRV02, JCG03, GCZ03, BBB04, HAW05, ATM07, JLF11, JLF12], and approaches on error mitigation [LV62, DP94, HTA94, SRK04, AT05, CM10, KPM09].

My dissertation is focused on error mitigation algorithms. Two major threats for the FPGA reliability come from hard errors arising from circuit processing and soft errors due to particle radiation. Section 2.3.1 and Section 2.3.2 present existing techniques addressing them, respectively.

2.3.1 Hard Errors Mitigation Algorithms

Targeting the hard errors due to manufacturing defects, the following techniques have been developed.

- *Locating and masking defects by circuit redundancy*

  Column-based redundancy technique, proposed in [DP94,HTA94], has been used in Altera's Stratix II FPGA [ALT06]. If one logic block in a column of logic blocks is found defective during testing of the device, the entire column is bypassed and its function is implemented by the redundant column. Besides redundant columns and rows, some fine-grained redundancy architectures were also proposed [DI00,YL05], where redundant routing resources are evenly distributed in the FPGA to tolerate defects. The aforementioned tolerance is transparent to FPGA users, and the same synthesis can be used for all chips of the same FPGA application. This manufacturer-masking approach lowers synthesis cost for massive production, but suffers from low defect coverage, large area overhead, and extra delay due to the bypass circuit. For example, only defective logic blocks within the same column are tolerated with one extra column as in Stratix II.

- *Circuit-wise synthesis*

  The technique has been applied to circuits with high defect rates, especially for nano-technologies [Nae05,JA07,BKC07]. Each defect is located, and then placement and routing is customized for each chip in order to work around defects. Circuit-wise synthesis is not suitable for massive production of one FPGA application. In addition, testing costs can be intolerably high.

- *Triple-modular redundancy (TMR)*

  Compared to the previous two approaches, TMR [LV62] does not require to detect and locate defects during synthesis. However, it has significant overhead on area, power and performance.

- *Multiple configurations*

  EasyPath by Xilinx, pre-develops multiple synthesis solutions for an FPGA application. During testing, each chip chooses a synthesis that can tolerate manufacturing defects for the particular application. Compared to circuit-wise synthesis, multiple configuration reduces testing and synthesis costs. Compared to TMR, multiple configurations reduce circuit overhead. However, the synthesis overhead increases.

  Most existing techniques suffer from either expensive testing overhead, excessive overhead on performance, power and area, long design time, or a low defect coverage rate.

2.3.2 Soft Errors Mitigation Algorithms

There are a number of studies on soft errors mitigation for FPGAs in the literature, including circuit, architecture, and synthesis based solutions.

- *Circuit solutions*

  Circuit based solutions create radiation hardened cells to shield against soft errors. For instance, in the space-grade Virtex-5QV FPGA released from Xilinx, the configuration memory has been implemented with radiation-hardened by design (RHBD) dual-node latches that provide nearly 1000$\times$ SEU hardness of the standard cell latches in the commercial version [XIL12b]. They come with considerable area, power, and cost overheads, and constraint themselves in applications with a high demand for the availability.

- *Architecture solutions*

  There are several solutions on the architecture level. Designers routinely apply circuit redundancy techniques (e.g., TMR [LV62]), to protect designs from SEUs. However, TMR has 3.2× area overhead and a performance cost of 10% [Roo04]. Samudrala et al. [SRK04] claimed that the area can be reduced by 60-70% using their selective TMR approach. Error checking and correction (ECC) [Tam06] is widely used in modern FPGAs, because its area overhead is lower than TMR. For instance, single error correct double error detect (SECDED) has the overhead of 8 bits per 64 bits of data (i.e., 13%). Nevertheless, ECC requires power and area overheads to scan and detect errors. Furthermore, errors fall into the scan interval are not tolerated. Mitra et al. [MSZ05] proposed the built-in soft error resilience (BISER) algorithm to use the on-chip design-for-testability resource for SEU protection. In summary, most architecture based solutions require area, power, performance, design time, or cost overhead.

- *Synthesis solutions*

  Several studies have demonstrated that the SEU issue can be mitigated by synthesis based approaches while minimizing the aforementioned overheads. Cong and Minkovich [CM10] reported that failure rates can be reduced by 12%, by choosing cuts with more don't cares during technology mapping with the help of their error analysis technique. Lee et al. [LFH10] proposed an in-place decomposition (IPD) algorithm to make the circuit more robust against soft errors. IPD introduces don't cares by the decomposed and duplicated subfunctions and the underutilized carry chains, and increases MTTF without changing the topology of the PLB network. The work claims to reduce failure rates by 76%. However, the algorithm does not tolerate defects on interconnect. Note that the preceding techniques [LFH10, CM10] consider SEUs on LUTs only, and their improvements for the reliability are significantly smaller when SEUs on interconnects are taken into consideration. Jose et al. [JHM10] proposed a rewiring algorithm for the robustness of interconnects. However, their interconnect SEU model assumed that each net has only one configuration bit, causing critical routes being used to replace non-critical ones. Most synthesis based techniques are based on time-consuming algorithms, e.g., binary decision diagrams (BDD), Boolean satisfiability (SAT), integer linear programming (ILP) [LFH10], or set of pairs of functions to be distinguished (SPFD) [JHM10]. In summary, most preceding synthesis based techniques require long synthesis time, and do not consider the SEU impact on interconnects.

CHAPTER 3

ROSE: Stochastic Resynthesis for FPGA Reliability

Targeting stochastic error rate reduction in the presence of both hard and soft errors while incurring minimal area and power overhead, we develop an formal method (i.e., stochastic satisfiability (SSAT)) based fault tolerant Boolean matching (FTBM), which exploits the flexibility of the look-up table (LUT) configuration in FPGAs to maximize the stochastic yield rate for a logic function. Using FTBM, we propose a robust resynthesis algorithm (ROSE) which maximizes stochastic yield rate for an entire circuit. Finally, we show that existing programmable logic block (PLB) templates for area-aware Boolean matching and logic resynthesis are not effective for fault tolerance, and we propose a new robust template with path re-convergence. Compared to the state-of-the-art academic technology mapper Berkeley ABC, ROSE using the proposed robust PLB template reduces the fault rate by 25% with 1% fewer LUTs, and increases mean time between failures (MTBF) by 31%, while preserving the optimal logic depth.

3.1 Introduction

Most of today's programmable logic device (PLD) synthesis flows target *nominal* designs. In this view, the low-level and uncertain physics of devices and transistors are abstracted into Boolean digital signals, 0 and 1, and a circuit is a *deterministic* function which maps input bits into output bits (and states). Logic synthesis optimizes the representation of these functions through Boolean reasoning. Deviations from the nominal behavior, e.g., through process variations and defects, are controlled at the testing level by discarding defective chips.

Unfortunately, as faults become more pronounced in emerging applications and technologies, such as permanent faults arising from circuit processing at nanometer scales or soft errors arising from high-energy particle hits, the deterministic view becomes limited. For CMOS circuits vulnerable to soft errors, these faults reduce the mean time between failures (MTBF). For future nano-circuits with more defective devices, they reduce the yield rate. This implies that logic design and synthesis flows must explicitly account for and tolerate faults.

Indeed, fault tolerance techniques have been studied extensively for FPGA [DH06]. Without considering dynamic re-configuration during runtime, the following techniques have been developed to tolerate manufacturing faults: (a) *Locating and masking faults by circuit redundancy* (e.g., column-based redundancy, proposed in [DP94, HTA94]); (b) *Chip-wise synthesis* proposed in [Nae05, JA07, BKC07]; (c) *Triple-modular redundancy (TMR)* [LV62]; (d) *Multiple configurations* (e.g., *EasyPath* by Xilinx). The detailed discussion about the aforementioned techniques is presented in Section 2.3.1. In summary, most existing techniques suffer from either expensive testing overhead, excessive overhead on performance, power and area, long design time, or a low fault coverage rate.

We take an alternate route toward fault tolerant designs. We propose *stochastic synthesis* for fault tolerance, where the presence of random faults is reflected in the logic synthesis algorithm. We model faults in LUT configurations and the faults in intermediate wires as random variables, following the probabilistic nature of faults, and the probability that a LUT configuration bit or an intermediate wire is defective is given as an input to our synthesis algorithm. Under these fault sources, the *fault rate* of a circuit is the percentage of primary input vectors under which the circuit does not produce the desired logic output values. Stochastic synthesis algorithms explicitly minimize fault rates, along with traditional metrics such as circuit area or delay.

As a particular example of stochastic synthesis, we propose ROSE, a RObust reSynthEsis algorithm, which minimizes the stochastic fault rate under random faults in FPGAs while incurring negligible area and performance overhead. ROSE exploits the flexibility in implementing a logic block by a programmable logic block (PLB) template (e.g., in selecting configuration bits), and rewrites logic blocks to minimize the stochastic fault rate. Unlike manufacturer-masking, ROSE does not need to locate faults by testing. Unlike chip-wise synthesis, ROSE uses the same design for different chips of an FPGA application for stochastic fault tolerance. It can be directly applied to tolerate faults in less critical tasks, such as internet routing switches and enterprise servers, to reduce MTBF. In addition, ROSE is orthogonal to the existing redundancy based fault-tolerant approaches [DP94, HTA94], and therefore it also can be used to further increase the robustness while reducing the overhead of existing techniques such as TMR and column based redundancy for FPGAs.

The core algorithmic idea in ROSE is that of *fault-tolerant Boolean matching* (FTBM). FTBM generalizes the Boolean matching problem [LSB05b,LSB05a] to the setting with stochastic faults. It takes as input a PLB $\mathcal{H}$, a Boolean function F, and the fault rates for the inputs and the SRAM bits of the PLB, and outputs either that F cannot be implemented by PLB $\mathcal{H}$, or the configuration of $\mathcal{H}$ which minimizes the probability that the faults are observable in the output of the PLB under all input vectors. We describe an algorithm for FTBM by a reduction to *stochastic satisfiability* (SSAT) [Pap85]. Since SSAT has a high computational cost (and state-of-the-art SSAT solvers are less developed than SAT solvers), we convert the SSAT problem into a sequence of deterministic Boolean satisfiability problems. While we apply FTBM in resynthesis, it can also be applied in various synthesis stages, such as technology independent optimization, technology mapping or post-mapping resynthesis.

ROSE applies FTBM on circuit logic blocks against a set of pre-defined *templates*, substituting the block with the minimum fault rate configuration of the template found by FTBM (provided this does not increase local logic depth or area). Orthogonally to FTBM, ROSE enhances robustness by a suitable choice of PLB templates. We evaluate some PLB templates for fault tolerance. First, we show that existing templates for area optimization are not effective for fault tolerance. Second, using the observation that reconvergence is a prime reason for don't cares, and that don't cares can occlude errors, we propose a new robust template with path reconvergence. Our template can be obtained by either configuring the existing full connection between LUTs within a cluster, or hardwiring selected connections between LUTs to reduce area.

On QUIP [ALT] benchmarks, ROSE (using the robust template) reduces the fault rate by 25% with 1% fewer LUTs, and increases MTBF (mean time between failures) by 31%, while preserving the optimal logic depth, when compared to the state-of-the-art FPGA technology mapping, Berkeley ABC mapper [Mis11].

The remainder of this chapter is organized as follows. Section 3.2 presents the motivation of the chapter, followed by preliminary definitions in Section 3.3. Sections 3.4 and 3.5 present the overall approach and FTBM, respectively. The experimental results are given in Section 3.6 and the chapter is concluded with future research directions in Section 3.7.

3.2 Motivation

In this section, we perform a preliminary experiment to demonstrate that different synthesis options have significant impact on defect-tolerance in FPGA technology mapping. We use an existing logic synthesis and technology mapping tool, Berkeley ABC [Mis11]. We conduct the following experiments on FPGAs with a set of MCNC benchmark applications. Each application is first synthesized by ABC using various combinations of different technology independent optimization algorithms, such as logic re-factorization (ABC alias “rf”), re-substitution (ABC alias “rs”), re-synthesis (ABC alias “rs”) and AIG re-writing (ABC alias “rw”). We apply 18 different synthesis options in our experiments to obtain 18 different synthesis solutions with equivalent functionality for each application. Each synthesized solution is then mapped to 4-input LUTs by ABC’s technology mapper with the command “fpga -K 4”.

We now randomly inject defects into each mapped solution, assuming a 1-bit configuration SRAM in an LUT can be defective (i.e., flipped) with the probability 0.01% independently of other bits. A chip is considered failed if equivalence checking (using ABC’s “cec”) w.r.t. the defect oblivious solution fails. The *yield loss* (the number of failed chips over total 10K chips) due the injected defects for each mapped solution is then counted based on Monte Carlo simulation with 10K times.

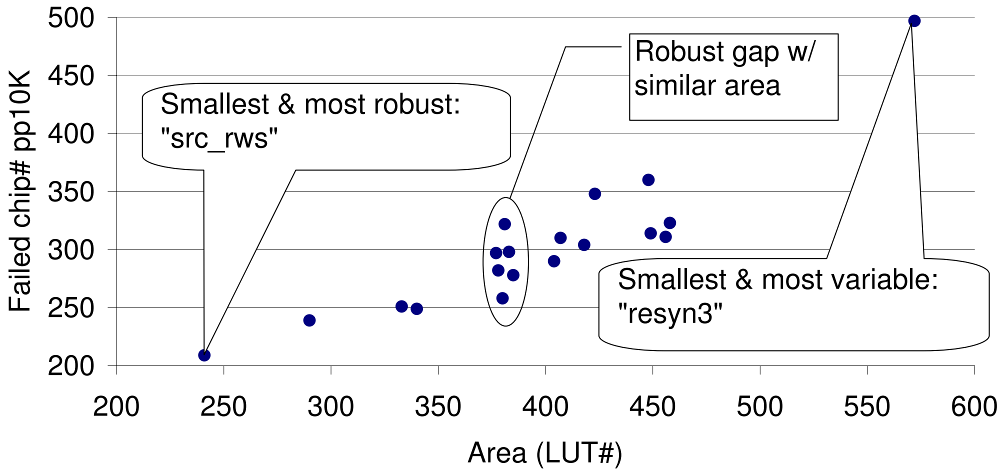


Figure 3.1: Yield loss vs. area (the number of LUT) for the MCNC application “t481” with 18 different logic synthesis options (1-bit defect rate: 0.01%)

Two MCNC applications, “t481” and “ttt2”, are chosen as the representatives to investigate the impact of logic synthesis to the yield. The simulation results

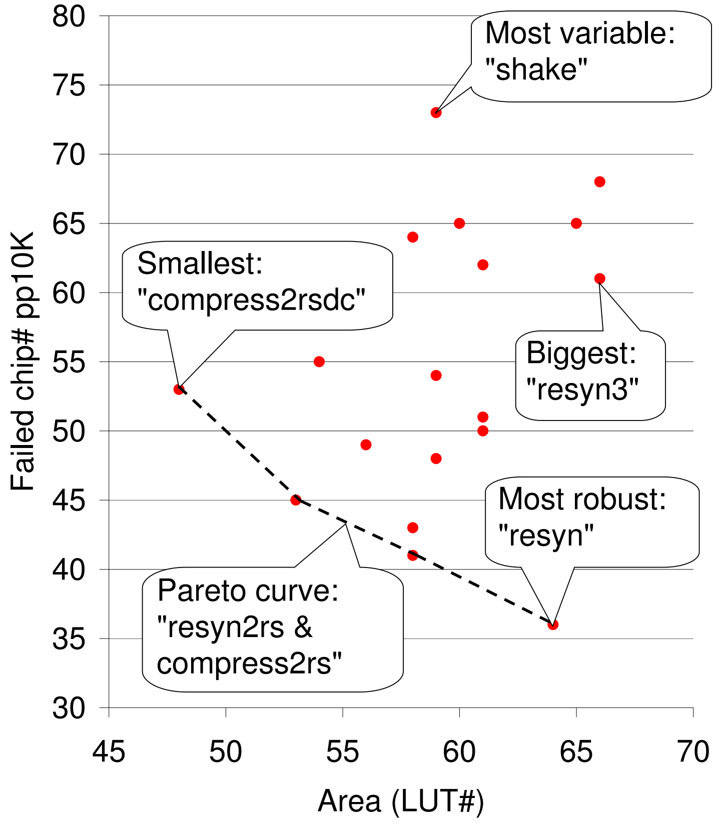


Figure 3.2: Yield loss vs. area (the number of LUT) for the MCNC application "ttt2" with 18 different logic synthesis options (1-bit defect rate: 0.01%)

as shown in Figure 3.1 and 3.2, where both figures plot the area (LUT#) vs. yield loss for all 18 different synthesis options for each application. The following interesting observations are obtained from the plots.

1. Different synthesis options result in designs with significantly different area and yield. In addition, one synthesis option will not consistently produce minimal area or minimal yield loss for different applications, which makes it impossible to directly optimize for robustness with the existing synthesis algorithms. For example, synthesis option "src_rws" results in minimal area and yield loss for application "t481" while "compress2rsdc" generates the smallest area and "resyn" produces the most robust design for application "ttt2".

2. The correlation between area and yield varies from one application to another. Figure 3.1 shows a trend that the solution with smaller area is generally less vulnerable to defects. However, as shown in Figure 3.2, where the smallest design (produced by synthesis option "compress2rsdc") and the biggest design (produced by synthesis option "resyn3") have similar defect rate while the most robust design (with option "resyn") is over 20% larger than the smallest design. More interestingly, a Pareto curve, including 4 area-yield trade-off points, is present in Figure 3.1.

3. Designs with similar area might have significantly different yield loss, as shown in both figures, which motivates us to reconfigure the LUTs to maximize the logic masking for defects and thus to increase the robustness zero or limited area overhead. In Figure 3.2, considering all seven solutions with about 58 or 59 LUTs, the yield loss varies from 41 pp10K to 73 pp10K (a 78% difference).

The above observations demonstrate the potential to mitigate defects during FPGA synthesis. In the rest of this chapter, we are concerned with finding an optimal synthesis algorithm which minimizes yield loss without sacrificing area optimality.

3.3 Preliminaries

3.3.1 Boolean Network

A PLB $\mathcal{H}$ consists of a network of interconnected logic devices with a set of input pins and an output pin. A *K-LUT* is a LUT with K inputs, one output, and 2^K LUT configuration bits.

A LUT-based *Boolean network* is represented using a directed acyclic graph (DAG) whose nodes correspond to LUTs and directed edges correspond to wires connecting the LUTs. The nodes in the lowest level of the DAG are called *circuit inputs* (CIs), which include the *primary inputs* (PIs) and the outputs of registers. The nodes in the highest level are called *circuit outputs* (COs), which include *primary outputs* (POs) and the inputs to registers.

A *fanin* (resp. *fanout*) *cone* of node n is a sub-network whose nodes can reach the fanin edges of n (resp. can be reached from the fanout edges of n). A *maximum fanout free cone* (MFFC) of node n is a subset of the fanin cone such that every path from a node in the subset to the CO passes through n. Informally, the MFFC of a node contains all the logic used exclusively by the node. When a node is removed or substituted, its MFFC can be removed.

A *cut* C of node n is a set of nodes of the network such that each path from a CI to n passes through at least one node in C; node n is called the *root* of *cut* C. A cut is K*-feasible* if the number of nodes in it does not exceed K. A *logic block* is a sub-network which covers all nodes found on the path from the outputs (called *root nodes* of the logic block) to the cut, including the roots and excluding the cut. In this chapter, we consider multi-input, single-output (MISO) logic blocks, but the proposed algorithm can be applied to multi-output, multi-output (MIMO) logic blocks [HSM08], as well.

3.3.2 Boolean Matching

Given a PLB $\mathcal{H}$ and a Boolean function F, the *Boolean matching problem* (BM) either maps function F to PLB $\mathcal{H}$ by describing an appropriate setting of the LUT configuration bits, or concludes that PLB $\mathcal{H}$ cannot implement function F. *Boolean matching* [CH01, BM97] is one of the most important sub-problems in logic synthesis and technology mapping for FPGAs.

The Boolean matching problem can be formulated as a (quantified) Boolean satisfiability problem in the following way [LSB05b, CM07, HSM07]. Consider a

PLB template $\mathcal{H}$ with inputs $x'_1, \cdots, x'_k$, output G, intermediate wires $z_1, \cdots, z_m$, and LUT configuration $c_1, \cdots, c_n$ as shown in Figure 3.3. Let F be a Boolean function of k inputs, given as a truth table.

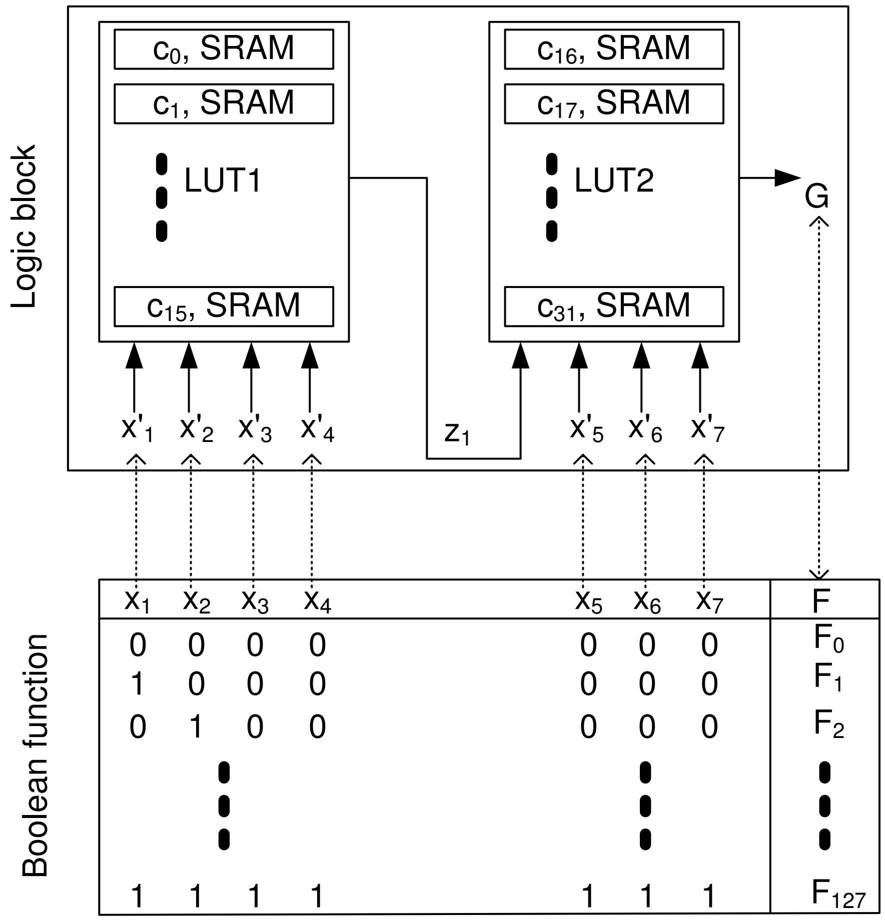


Figure 3.3: Illustration of FPGA Boolean matching

We can write a set of Boolean constraints that define each internal and output wire of $\mathcal{H}$ in terms of its inputs (see, e.g., [LSB05b, HSM07]). For example, the internal wire z_1 in Figure 3.3 can be defined as

$$(\overline{x'_1} \wedge \overline{x'_2} \wedge \overline{x'_3} \wedge \overline{x'_4} \rightarrow (z_1 \leftrightarrow c_0)) \wedge \cdots \wedge$$
$$(x'_1 \wedge x'_2 \wedge x'_3 \wedge x'_4 \rightarrow (z_1 \leftrightarrow c_{15}))$$

Let $\Psi(\mathcal{H})$ be the conjunction of constraints defining each wire of $\mathcal{H}$.

Similarly, the truth table for function F can be expressed as a set of constraints between the input variables $x_1, \cdots, x_k$ and the output F:

$$\begin{aligned}\Psi(F) \quad = \quad & (\overline{x_1} \wedge \overline{x_2} \wedge \cdots \wedge \overline{x_k} \rightarrow F_0) \wedge \\ & (x_1 \wedge \overline{x_2} \wedge \cdots \wedge \overline{x_k} \rightarrow F_1) \wedge \cdots \wedge \\ & (x_1 \wedge x_2 \wedge \cdots \wedge x_k \rightarrow F_{2^k-1}) \end{aligned} \tag{3.1}$$

where $F_i = F$ if $F(i) = 1$, otherwise, $F_i = \overline{F}$.

The Boolean matching problem for $(\mathcal{H}, F)$ can be then expressed as the quantified Boolean formula problem that asks, does there exist some setting of the LUT configuration $c_1, \cdots, c_n$ such that for all inputs $x_1, \cdots, x_k$, the output G of $\mathcal{H}$ is equivalent to F? Formally, we ask:

$$\exists c_1 \cdots c_n \forall x_1 \cdots x_k \exists z_1 \cdots z_m \,.\, \Psi(\mathcal{H}) \wedge \Psi(F) \wedge (G \leftrightarrow F) \tag{3.2}$$

By replicating the formula for each possible valuation to the bits $x_1 \cdots x_k$, we reduce the quantified formula to an (existential) satisfiability problem. Each satisfying assignment gives an instantiation of the LUT configuration bits that implement the same function F. Most existing work for Boolean matching is based on function decomposition [CH01] or on canonicity and Boolean signatures [BM97]. These approaches are limited by the input size of the functions they can handle and the flexibility required by certain CAD tasks, such as FPGA architecture evaluation [LSB05a]. Recently, a SAT-based approach [LSB05b], with high flexibility and the good trade-off between space and time complexity, has been proposed to solve Boolean matching. This was improved by [SVB06, CM07] with up to 13x speedup, and was further improved by [HSM07] with 100x speedup.

3.3.3 Fault Model and FTBM

In the presence of faults in the LUT configurations or intermediate wires between PLBs, we extend the Boolean matching algorithm in the following way. We model faults in LUT configurations and the faults in intermediate wires as random variables, and assume that the probability that a LUT configuration bit or an intermediate wire is defective is known. Under these fault sources, the *fault rate* of a circuit is the percentage of primary input vectors under which the circuit does not produce the desired logic output values. Based on the above fault modeling, we formulate *fault-tolerant Boolean matching* (FTBM) as follows.

Definition 1. ***FTBM*** *takes as input a PLB $\mathcal{H}$, a Boolean function F, and the fault rates for the input pins (representing intermediate wires between PLBs) and the configuration bits of the PLB, and outputs either that F cannot be implemented by PLB $\mathcal{H}$, or the configuration of $\mathcal{H}$ which minimizes the probability, over all input vectors, that the faults are observable in the output of the PLB.*

Note that faults at PLB input pins result from faults of the upstream logic and wires between PLBs. While we assume single fault in our experiments, our algorithm for FTBM (described in Section 3.5) allows multiple faults to occur simultaneously.

3.3.4 Resynthesis for Fault Tolerance

Resynthesis [Mic91, MCB05, MSB91, BM07] is a technique that rewrites circuit structures while maintaining the functionalities of transition and output functions to reduce area. It can be performed simultaneously with technology mapping or as a post-mapping optimization. The simultaneous approaches perform logic resynthesis, such as Boolean decomposition of logic functions [LWG97,MWK03], during the mapping process. Since they explore a large solution space, the simultaneous approaches tend to be time-consuming, limiting their use to small designs. To handle large designs, resynthesis is usually performed after technology mapping for area recovery. Recently, [LSB05b, CM07, HSM08] proposed a combinational resynthesis based on Boolean matching [BM97] for FPGA area reduction. The resynthesis is performed by mapping logic blocks extracted from a circuit against a library set of logic blocks (e.g., hard-wired LUTs) and replacing them with a functionally equivalent logic block if area can be reduced.

While resynthesis guarantees a functionally equivalent circuit, the fault rate of the resynthesized circuit can be significantly different from the original one. We demonstrate this by examples in Figure 3.1 and Figure 3.2 in Section 3.2, which show the fault rates vs. area (the number of LUTs) for 18 resynthesis solutions obtained using different options to ABC [Mis11]. Notice that the same application may be implemented using multiple distinct configuration settings with different logic masking, resulting in significantly different fault tolerance but with similar area. Hence, we propose a *robust* resynthesis algorithm to simultaneously reduce circuit area and fault rate.

3.4 Robust Resynthesis

The fault rate of a circuit is impacted by both the synthesis algorithm and the topological structure of the implementation. In this section, we first describe the overall flow of our proposed robust resynthesis algorithm ROSE, and then present a robust PLB template which enables better fault tolerance with ROSE.

3.4.1 Overall Algorithm of ROSE

Our procedure takes an application mapped to K-LUTs and scans the combinational portion of the circuit in topological order from primary inputs to primary outputs. In the course of scanning, new logic blocks are generated by combining the logic blocks at the input LUTs. Each logic block is mapped against one or more pre-defined PLB templates; if a mapping with the minimal fault rate is found by FTBM, the logic block can be substituted by the PLB template. However, any substitution that increases the local logic depth or area is discarded. This ensures

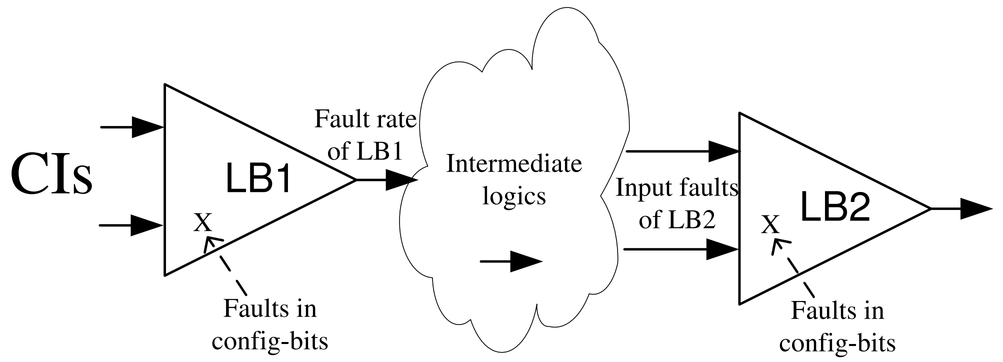


Figure 3.4: Propagation of faults in ROSE, where input faults of LB2 are resulted from the output fault in LB1.

that the logic depth and area does not increase. In our implementation, only MFFCs are considered as candidates for mapping.

As the resynthesis of a logic block will change the fault rate of its output and therefore change the fault rates observable by the inputs of the downstream network, ROSE processes all MFFCs in a topological order (from CIs to COs) to guarantee that the input fault rates of a logic block have been correctly updated before the block is resynthesized. To calculate the fault rate for a logic block, both faults in LUT configurations and the inputs of the block need to be considered. After resynthesis, we can obtain the fault rate of the block output and need to update the fault rates for all downstream intermediate pins under the fanout cone of the block output (see Figure 3.4).

3.4.2 Robustness of PLB Templates

Besides an effective robust resynthesis algorithm, it is also important to find an effective PLB template for fault tolerance, because different templates may have significantly different capability of carrying fault tolerance and therefore they can pre-determine the potential of the effectiveness of FTBM.

We consider Boolean functions with up to 10 inputs. According to [LSB05b], there are three possible PLB templates with no-more-than three 4-LUTs to implement a Boolean function with up to 10 inputs (see Figure 3.5 (a),(b) and (c)). The inherent disadvantage of these area efficient PLB templates is the lack of opportunities to place don't cares, which are the major source of logic masking and fault mitigation. Inspired by a well-known observation that re-convergence is a prime reason for don't cares, we propose a new PLB template, R-PLB, as shown in Figure 3.5 (d), which requires four 4-LUTs and forms re-convergent paths from input to output.

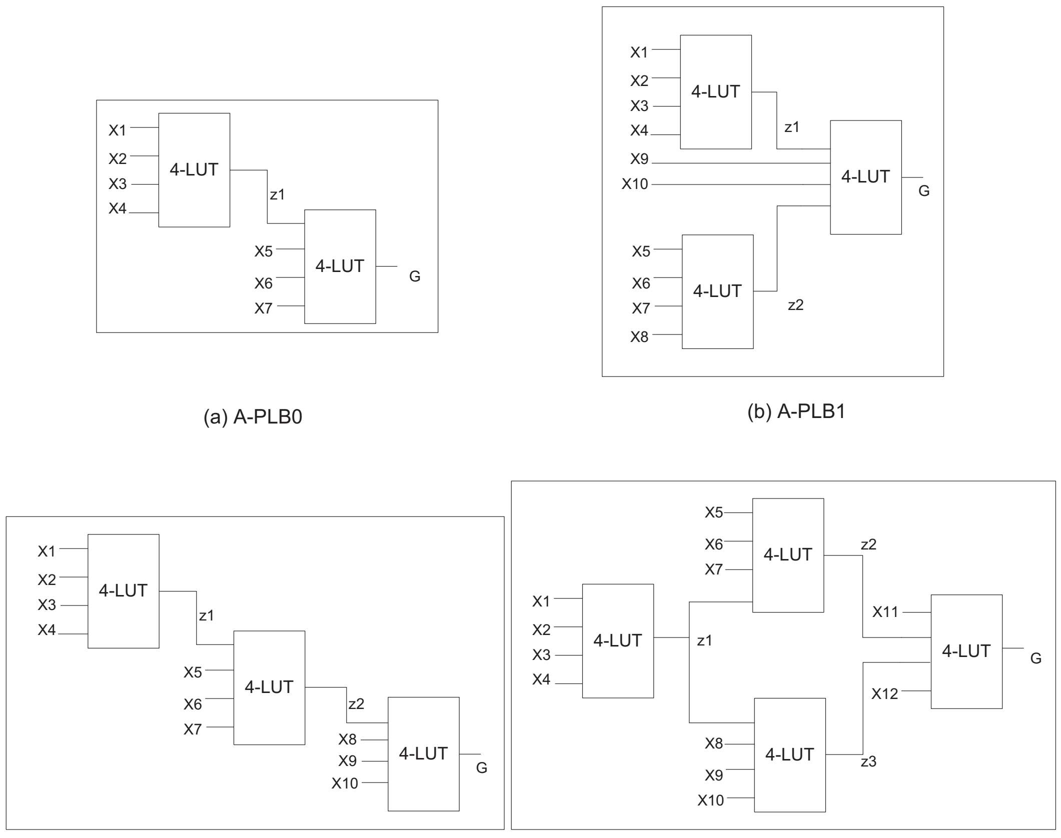


Figure 3.5: Area efficient PLB templates (a),(b) and (c), and the proposed robust template (d)

The logic don't cares include *satisfiability don't cares* (SDCs) due to the fact that some combinations are not produced as the inputs of the node, and *observability don't cares* (ODCs) due to the fact that under some conditions the output value of the node does not matter (i.e., is controlled by certain input combinations) [MB05b]. Figure 3.6 shows examples of don't cares in R-PLB. For simplicity, we use 2-LUT to replace 4-LUT. In Figure 3.6(a), LUTs z_2 and z_3 implement 2-AND and 2-OR, respectively. Gate z_2 and z_3 form a SDC for LUT G because input vector $z_2 = 1 \wedge z_3 = 0$ will never happen due to the path reconvergence, therefore the faults in configuration bit 10 in LUT G will not affect the output; In Figure 3.6(b), LUTs z_2 and z_3 both implement 2-OR. $x_3 = 1 \wedge x_4 = 1$ is the control signal of gate z_2 and z_3, which masks the output of LUT z_1, i.e., makes z_1 an ODC for the output, and therefore any faults in configuration bits in LUT z_1 will not affect the output. In Section 3.6, we will experimentally show the effectiveness of R-PLB for fault tolerance.

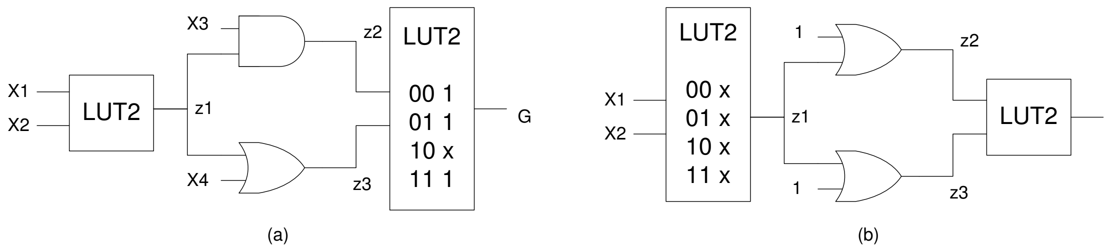


Figure 3.6: Examples of logic masking in R-PLB.

3.5 FTBM Algorithms

We now describe an algorithm for FTBM, which is the core of ROSE, and discuss implementation issues. Recall the CNF-encoding procedure described in Section 3.3.2, after solving (3.2), a set of LUT configurations $c_1, \cdots, c_n$ will be returned by the SAT solver if F can be implemented by $\mathcal{H}$. There might exist multiple distinct implementations (i.e., different configurations) for $\mathcal{H}$ all of which implement F.

In fact, we can obtain partial or even all feasible configurations by iteratively adding the negation of previously obtained configurations into the CNFs and solving an augmented SAT problem. For each of these feasible configurations, $\mathcal{C} = (c_1^{'}, \cdots, c_n^{'})$, we evaluate the fault rate at the output of this logic block under this configuration setting. The configuration, $\mathcal{C}^*$, which results in the minimal fault rate is chosen as the candidate for mapping or resynthesis.

3.5.1 Fault Rate Calculation

The main step of FTBM is an algorithm for fault rate calculation. We now show how fault rate calculation can be reduced to stochastic satisfiability (SSAT) [Pap85], a generalization of Boolean satisfiability where some variables may be "randomly quantified" in addition to variables that are existentially or universally quantified. For example, the formula

$$\exists x_1, \Re x_2, \forall x_3, \ldots, \exists x_{n-1}, \Re x_n.(E\varphi(x_1, \ldots, x_n) \geq \beta)$$

asks whether there exists a value for x_1 such that *for random* values of x_2 (chosen from a given distribution), for all values of x_3, ... there exists a value of x_{n-1} such that for random values of x_n the *expected* value $E\varphi$ that the Boolean formula φ is satisfied under the variable assignment is at least β. SSAT has been studied in the AI community to reason about planning under uncertainty and belief networks [Lit99], and detailed complexity bounds are known [LMP01]. It is known that stochastic satisfiability can be solved using DPLL style algorithms [Lit99], and many heuristics, such as non-chronological backtracking, can be applied in this context as well [MB05a]. However, SSAT tools are not as mature as SAT solvers in practice.

For a logic block, we assume two (independent) sources of faults: defective input bits of the logic block (resulting from the faults of the upstream logic and wires between PLBs) and defective LUT configurations inside the block. Assume the fault rate of input bit x'_i is P_i, i.e., that with probability P_i, the ith input in the mapped circuit is the opposite of the ith input of the function F. For each i, we introduce a Boolean variable p_i, where $p_i = 1$ with probability P_i and $p_i = 0$ with probability $1 - P_i$ and the constraint

$$p_i \leftrightarrow (x'_i \neq x_i) \tag{3.3}$$

to indicate that p_i is 1 iff the input bit x'_i is not equal to x_i. Similarly, assume that the fault rate of the LUT configuration c_i is D_i, i.e., with probability D_i, the ith LUT configuration bit in the logic block has a value opposite to the nominal LUT configuration bit. We introduce a Boolean variable d_i that is 1 with probability D_i, and add the constraint

$$d_i \leftrightarrow (c'_i \neq c_i) \tag{3.4}$$

to indicate that the ith LUT configuration bit c'_i is different from the correct value c_i iff d_i is 1.

Then, given a threshold β, the Boolean matching problem under random faults in the input bits and the LUT configuration of the logic block is reduced to the stochastic satisfiability instance:

$$\begin{aligned}
&\exists c_1, \cdots, \exists c_n, \Re p_1, \cdots, \Re p_k, \Re d_1, \cdots, \Re d_n, \exists c'_1, \cdots, \exists c'_n, \\
&\forall x_1, \cdots, \forall x_k, \exists x'_1, \cdots, \exists x'_k, \exists z_1 \cdots \exists z_m, \exists G, \exists F \\
&E\{\Psi(\mathcal{H}) \wedge \Psi(F) \wedge (G \leftrightarrow F) \wedge \\
&\bigwedge_{i=1,\cdots,n} d_i \leftrightarrow (c'_i \neq c_i) \wedge \bigwedge_{i=1,\cdots,k} p_i \leftrightarrow (x'_i \neq x_i)\} \geq \beta
\end{aligned} \tag{3.5}$$

where $\Psi(\mathcal{H})$ and $\Psi(F)$ are the constraints that define the internal and output signals of $\mathcal{H}$ and the truth table F, respectively. The probabilities $P(d_i = 1) = D_i$ are given as inputs to the problem, and $P(p_i = 1) = P_i$ are obtained from the upstream logic block. If the above SSAT problem is satisfiable, then the choice of the LUT configuration $c_1, \ldots, c_n$ ensures that the probability that the circuit implements F even when the LUT configuration and inputs are flipped is at least

β. By a binary search on β, we can find the maximal probability that the defective circuit implements the function F for this choice of the LUT configuration.

The above formulation (3.5) requires *all* min-terms of G and F are identical under all input vectors $x_1, \cdots, x_k$, which is too restrictive in practice. Therefore, we relax the definition of fault rate as the percentage of min-terms produced by G that are not equal to the corresponding min-term in function F. Formally, suppose logic block $\mathcal{H}$ with output G and the LUT configuration set to $c_1, \ldots, c_n$ implements a Boolean function $F(x_1, \cdots, x_k)$. The fault rate of logic block $\mathcal{H}$ under the input and the LUT configuration faults is defined as

$$DF(\mathcal{H}) = \frac{1}{2^k} \sum_{x_1,\cdots,x_k=0,\cdots,0}^{1,\cdots,1} DF_{\text{min-term}(x_1,\cdots,x_k)}$$

where $DF_{\text{min-term}(x_1,\cdots,x_k)}$ is the probability that G and F have different values when the input is fixed at $(x_1, \ldots, x_k)$ and faults occur randomly according to P_i and D_i. This probability can be computed by maximizing β in the SSAT formula:

$$\begin{aligned}
&\Re p_1, \cdots, \Re p_k, \Re d_1, \cdots, \Re d_n, \\
&\exists c'_1, \cdots, c'_n, \exists x'_1, \cdots, \exists x'_k, \exists z_1 \cdots \exists z_m, \exists G, \exists F \\
&E\{\Psi(\mathcal{H}) \wedge \Psi(F) \wedge (G(x_1, \cdots, x_k) \leftrightarrow F(x_1, \cdots, x_k)) \wedge \\
&\bigwedge_{i=1,\cdots,n} d_i \leftrightarrow (c'_i \neq c_i) \wedge \bigwedge_{i=1,\cdots,k} p_i \leftrightarrow (x'_i \neq x_i)\} \geq \beta
\end{aligned} \tag{3.6}$$

Note that the bits $c_1, \ldots, c_n$ and $x_1, \ldots, x_k$ are assumed to be fixed in the above formula. Finally, we have

$$DF_{\text{min-term}(x_1,\cdots,x_k)} = 1 - \hat{\beta},$$

where $\hat{\beta}$ is the maximal probability of satisfying formula (3.6).

3.5.2 Reduction to Deterministic SAT

While there are tools to solve SSAT directly [Lit99], we found that their runtimes are too long to be applied directly to our problem. For example, it takes over four hours on a Xeon 1.9GHz workstation to solve a SSAT instance with 16 random variables, 14 universal variables and 2K clauses using the implementation from [Lit99]. Instead, we solve Equation (3.6) iteratively using a deterministic SAT solver (miniSAT [ES] in our experiments), by enumerating satisfying assignments and evaluating the expectations according to the probability distributions of the randomly quantified variables.

We first express (3.6) as a deterministic SAT problem by changing the random quantifiers before p_i's and d_i's to existential quantifiers:

$$\begin{aligned}
&\exists p_1, \cdots, \exists p_k, \exists d_1, \cdots, \exists d_n,\\
&\exists c'_1, \cdots, \exists c'_n, \exists x'_1, \cdots, \exists x'_k, \exists z_1 \cdots \exists z_m, \exists G, \exists F\\
&\{\Psi(\mathcal{H}) \wedge \Psi(F) \wedge (G(x_1, \cdots, x_k) \leftrightarrow F(x_1, \cdots, x_k)) \wedge\\
&\bigwedge_{i=1,\cdots,n} d_i \leftrightarrow (c'_i \neq c_i) \wedge \bigwedge_{i=1,\cdots,k} p_i \leftrightarrow (x'_i \neq x_i)\}
\end{aligned} \tag{3.7}$$

Then, we solve an ALL-SAT problem for (3.7), obtaining all satisfiable assignments for p_i's and d_i's:

$$(p_1^1, \cdots, p_k^1, d_1^1, \cdots, d_n^1), \cdots, (p_1^m, \cdots, p_k^m, d_1^m, \cdots, d_n^m)$$

The maximal probability of satisfying (3.6) can be calculated by using the following formula:

$$\hat{\beta} = \frac{1}{2} \sum_{i=1}^{m} \left(\prod_{j=1}^{k} Pr(p_j = p_j^i) \cdot \prod_{j=1}^{n} Pr(d_j = d_j^i) \right)$$

assuming the independence of each fault [1] and the probabilities P_i and D_i. Note that P_i denotes the probability that either $(x_i = 0 \wedge x'_i = 1)$ or $(x_i = 1 \wedge x'_i = 0)$. If we distinguish between these two cases assuming that they are equally probable, we have that $(x_i = 0 \wedge x'_i = 1)$ with the probability $(P_i)/2$ and $(x_i = 1 \wedge x'_i =$

[1] Although independence between faults is assumed here for the reduction from SSAT to deterministic SAT, such reduction can be done for multiple faults with given correlation. Therefore, the SSAT-based formulation and algorithm can be extended to handle multiple faults.

0) also with the probability $(P_i)/2$. Therefore we have to distinguish how the corresponding variables are assigned and to use $P_i/2(D_i/2)$ in our calculation.

An ALL-SAT problem (e.g., 3.7) can be solved by iteratively adding the negation of the previously satisfiable solutions into the CNFs, which is computationally expensive, and we exploit the structure of the problem to speed up the fault rate calculation. We note that the probabilities P_i and D_i are usually very small. So we rule out certain combinations of p_i's and d_i's that have very low probability of occurrence. For example, we can restrict the search to assignments in which at most a small number of p_i's and d_i's are one simultaneously. When the fault rate of single bit is low, this is a nice approximation that may slightly reduce the total fault rate but with significant speedup of the reasoning runtime. In our experiments, we shall assume there is a single faulty bit.

3.6 Experimental Results

3.6.1 Evaluation Based on Boolean Functions

We have implemented ROSE in C++ in the OAGear package [XPC05]. We use miniSAT2.0 [ES] as the SAT engine in FTBM. All experimental results are collected on a Ubuntu workstation with 2.6GHZ Xeon CPU and 2GB memory. We test our algorithms on QUIP benchmarks [ALT], where each benchmark is mapped by Berkeley ABC mapper [Mis11] with 4-LUTs. We assume that one and only one bit of the LUT configuration in the mapped FPGAs is defective. The fault rate of the chip is the percentage of the input vectors that produce the defective outputs, and it is calculated by Monte Carlo simulation with 20K iterations where one bit fault is randomly injected in each iteration.

We evaluate the effectiveness of the three templates for fault tolerance. We first extract 3000 9-input Boolean functions from QUIP benchmarks by enumerating 9-

feasible cuts. Without considering input faults, we perform FTBM to map these 9-input Boolean functions against A-PLB1, A-PLB2, and R-PLB, respectively. The configurations with minimal or maximal fault rate can be obtained by using two versions of FTBM algorithms, i.e., FTBM$^+$ and FTBM$^-$, where FTBM$^-$ returns the configurations with the maximal fault rate while FTBM$^+$ returns the ones with the minimal fault rate. We compare R-PLB with A-PLB1 and A-PLB2 in terms of (a) the minimal fault rate that can be achieved by these templates, and (b) the gap of the fault rates (i.e., the difference between the minimal and maximal fault rates) under all feasible configurations. Due to the space limit, we only show the comparison results between R-PLB and A-PLB2 in Figure 3.7. In these plots, only those Boolean functions that can be implemented by both templates are taken. As shown in the plots, the minimal fault rates achievable by R-PLB are generally less than those achievable by A-PLB2. In addition, the solution space, in terms of the gap of the fault rates that can be produced by all feasible configurations of template R-PLB, is generally wider than that of template A-PLB2, which indicates that there exists more flexibility to place don't cares in R-PLB. Similar observations are obtained by comparing R-PLB and A-PLB1. With these observations, we consider R-PLB as a more effective template for fault mitigation. Nevertheless, all other templates, A-PLB0, A-PLB1, and A-PLB2, can be tested by FTBM during technology mapping or resynthesis to trade-off area and fault rate.

3.6.2 Evaluation Based on Benchmark Circuits

Under the same setting in Section 3.6.1, we now evaluate the effectiveness of ROSE for fault tolerance. All ABC-mapped QUIP benchmarks are resynthesized by ROSE using area-efficient PLB templates (Figure 3.5 (a)-(c)) and the robust PLB template (Figure 3.5 (d)). In resynthesis using area-efficient templates, each of the three templates is examined and the configuration with the maximal fault rate is returned. The logic depths are preserved in all resynthesis algorithms. The results are summarized in Table 3.1 and Table 3.2.

We first show the gap between the minimal fault rate and the maximal fault rate that can be achieved by ROSE. The FTBM$^+$ and FTBM$^-$ mentioned in the previous sub-section are used in ROSE to obtain the minimal and maximal fault rates, respectively. As shown in column "gap" in Table 3.1, ROSE using FTBM$^+$ consistently returns results with smaller fault rates compared to ROSE using FTBM$^-$, and the gap is up to 2.42%. Note that ROSE only maximizes the local logic masking, i.e., the best logic masking introduced by ROSE may not always be the most effective to the primary outputs. The above observation indicates our approach in ROSE is a high quality heuristic in terms of global optimization. In the following experiments, we always use FTBM$^+$ in ROSE. In addition, ROSE/R using the robust template consistently obtains a bigger gap than ROSE/A using area-efficient templates. This demonstrates that the robust template does offer a higher potential for fault rate reduction.

Moreover, Table 3.1 compares the fault rates resulted from ABC (sub-column "ABC"), ROSE using area-efficient templates (sub-column "ROSE/A") and ROSE using robust template (sub-column "ROSE/R"), respectively. Compared to the mapping by ABC, our ROSE using the robust template reduces fault rate by 25% (1.06% vs. 0.80%) on average. Compared to ROSE using area-efficient templates, ROSE using the robust template reduces fault rate by 14% (0.80% vs. 0.93%) on average.

As shown in column "area" of Table 3.2, our ROSE using area-efficient templates and the robust template reduces area by 10% and 1%, respectively, compared to ABC mapping results. Since R-PLB has larger area compared to area-efficient templates, ROSE/R results in 11% larger area than ROSE/A.

In addition, Table 3.2 compares the runtime of the deterministic SAT-based resynthesis using area efficient templates [HSM08]. The proposed ROSE using two different template sets. It shows that ROSE using area efficient templates and ROSE using the robust template has 3X and 10X runtime overhead, respectively, compared to the deterministic resynthesis. The ROSE using the robust template is slower than the one using area efficient templates because the robust template contains more LUTs and FTBM requires more runtime to solve the SAT problem.

3.6.3 Estimation of MTBF

We now estimate the mean time between failure (MTBF) obtained by different resynthesis algorithms for industrial FPGAs. As suggested by [MER05], the MTBF can be estimated by the following formula:

$$\begin{aligned} \text{MTBF} &= 10^9/(24 \cdot 365)\ \text{FIT}_{\text{total}} \\ \text{FIT}_{\text{total}} &= 100 \cdot R_{\text{vulnerability}} \cdot R_{\text{intrinsic error}} \\ R_{\text{intrinsic error}} &= \text{Area} \cdot R_{\text{FIT}} \end{aligned} \tag{3.8}$$

where $R_{\text{vulnerability}}$ is the vulnerability factor, which is the fraction of faults that become errors, and can be estimated by the mean of the fault rates in Table 3.1, and $R_{\text{intrinsic error}}$ is the intrinsic error rate, which is proportional to the area (SRAM bits number) and the raw FIT rate[2] for a single bit, R_{FIT}. Typically, R_{FIT} is 0.001-0.01 FIT/bit and we use 0.01 FIT/bit in our estimation.

We use 330,000 4-LUTs as the typical industrial FPGA size to estimate the MTBF obtained by our ROSE. As shown in Table 3.3, ROSE using the robust template increases MTBF by 31% (27.15 years vs. 20.66 years) compared to ABC. Note that the MTBFs in our experimental results match the typical server system reliability goals as reported in [MER05], which validates our experimental settings. Note that the purpose of this comparison is to provide a sanity check for the effectiveness of our approach; clearly there exist many differences between FPGAs and server systems, but we do expect "ball-park" parameter values for

[2]One FIT is one failure in a billion hours.

MTBF calculations to be similar.

3.7 Conclusions and Future Work

We are encouraged by the initial success of ROSE in minimizing the stochastic fault rate while preserving optimal logic depth and minimally impacting area. On QUIP benchmarks, ROSE reduced the fault rate by 25% with 1% fewer LUTs, and increased MTBF by 31%, while preserving the optimal logic depth, when compared to ABC [Mis11]. We believe that our study of robust FPGA resynthesis provides a first step toward a general methodology for stochastic synthesis. In particular, there are several open directions.

- Our experiments assume single fault, but the proposed algorithm can deal with multiple uncorrelated faults, which will be first explored in future work. In addition, we will extend the proposed algorithm to consider given correlations between faults.

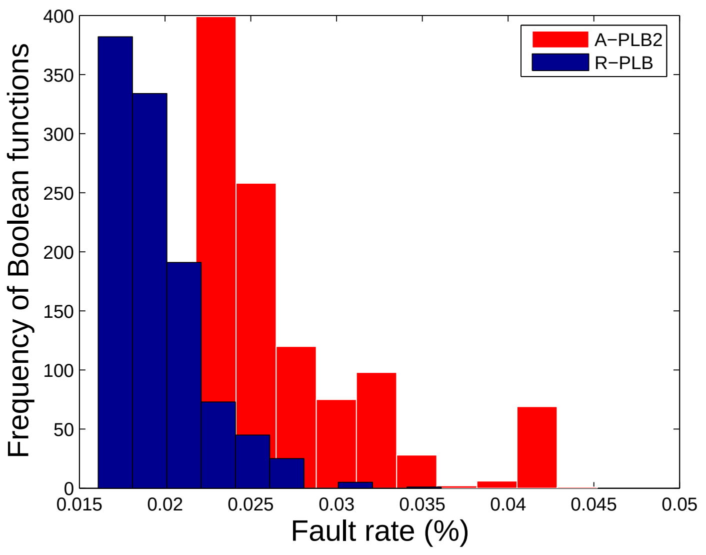
- To cope with the runtime overhead of the SAT-based Boolean matching, we will study more efficient alternatives for Boolean matching, e.g., building a hybrid algorithm combining SAT and Boolean decomposition [MBC08], which enables us to perform the resynthesis on logic blocks with wider inputs and leading to optimization for robustness from a more global point of view. In addition, potential advance of SSAT solvers will also provide more optimization power of our resynthesis.

- As the essence of the proposed stochastic synthesis is to take advantage of don't cares in a logic network, we will explore the don't cares explicitly for robust resynthesis. For example, one can first calculate the complete don't cares [MB05b] for a logic block, and then explore all possible mapping solutions considering these don't cares and select the one which maximizes logic masking to prevent the fault propagation. More generally, one can take advantages of the existing flexibilities in the network, e.g., Boolean relations [BS89], Sets of Pairs of Functions to be Distinguished (SPFDs) [YSN96], and sequential flexibilities [CKM08]. These may lead to more efficient algorithms and in turn more globally optimized solutions.

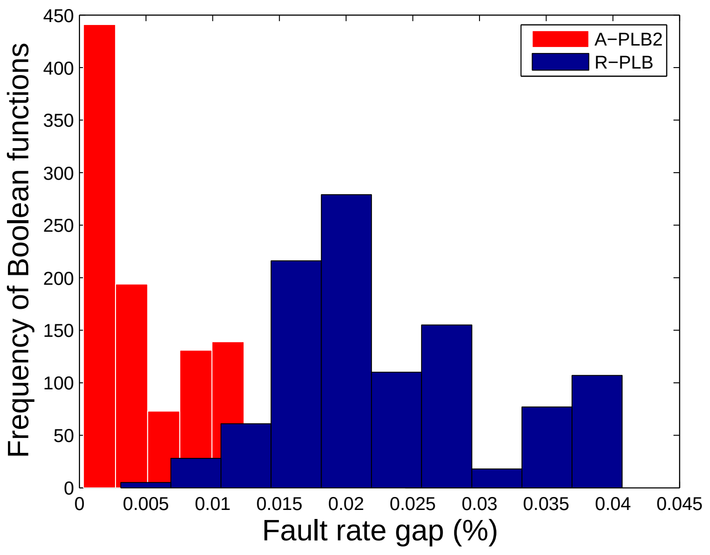
- By introducing additional logic masking, defect-aware logic synthesis algorithms such as ROSE can make verification and silicon debugging tasks difficult. In the future, it is necessary to address the tradeoff between logic masking and testability or verifiability. One possible solution is to provide proofs of correctness of transformations [KLG08].

- In the longer term, we shall investigate how other logic synthesis and optimization algorithms can be made fault-tolerant by exploiting redundancy or flexibility in the solution space. Finally, we shall investigate how fault tolerance at the logic synthesis level interacts with algorithm-level fault tolerance [Neu56].

- Our algorithm applies to standard cell-based circuits as well. There is some existing work on fault-tolerant logic synthesis for standard cell designs. For example, [NS04] developed a critical-area driven technology mapping, and [KPM07] applied logic redundancy and structural restructure to mask soft errors based on a fast simulation. Our work extends these ideas to an explicit formulation of stochastic synthesis.



(a) Achievable minimal fault rate



(b) Gap between minimal and maximal fault rates

Figure 3.7: Comparison between template R-PLB and template A-PLB2

benchmarks	gap		fault rate		
	ROSE/R	ROSE/A	ABC	ROSE/A	ROSE/R
barrel64	2.42%	0.64%	2.09%	1.82%	1.63%
fip_cordic_cla	0.62%	0.35%	1.25%	1.07%	0.93%
fip_cordic_rca	0.54%	0.31%	1.17%	1.03%	0.84%
mux8_128bit	0.30%	0.10%	0.97%	0.94%	0.67%
nut_000	0.51%	0.38%	0.97%	0.83%	0.63%
nut_002	0.44%	0.25%	0.71%	0.60%	0.48%
nut_004	0.32%	0.23%	0.69%	0.58%	0.50%
oc_ata_ocidec1	0.52%	0.14%	0.72%	0.65%	0.52%
oc_ata_ocidec2	0.66%	0.18%	0.88%	0.79%	0.64%
oc_ata_v	0.38%	0.19%	0.55%	0.47%	0.40%
oc_cordic_p2r	2.06%	0.64%	3.35%	3.09%	3.09%
oc_correlator	0.13%	0.20%	0.67%	0.64%	0.59%
oc_dct_slow	0.37%	0.20%	0.55%	0.46%	0.36%
oc_des_area_opt	0.83%	0.47%	1.26%	1.08%	0.93%
oc_des_des3area	1.25%	0.52%	2.04%	1.89%	1.44%
oc_i2c	0.37%	0.22%	0.62%	0.52%	0.46%
oc_rtc	0.60%	0.30%	0.95%	0.79%	0.71%
oc_sdram	0.65%	0.28%	0.76%	0.64%	0.60%
os_sdram16	0.78%	0.35%	0.97%	0.83%	0.74%
geomean	0.58%	0.30%	1.06%	0.93%	0.80%
normalized mean			1	0.87	0.75

Table 3.1: Comparison of fault tolerance between ABC and ROSE using robust and area-efficient PLB templates

benchmarks	area (LUT#)			runtime (min)		
	ABC	ROSE/A	ROSE/R	[HSM08]	ROSE/A	ROSE/R
barrel64	1862	1414	1734	1.55	9.22	16.77
fip_cordic_cla	1044	823	1027	0.7	0.78	7.10
fip_cordic_rca	983	789	970	0.58	0.55	5.88
mux8_128bit	898	770	898	0.23	0.25	149.03
nut_000	927	912	926	2.23	182.03	6.38
nut_002	652	645	652	3.03	4.2	4.80
nut_004	618	550	615	0.35	1.1	8.27
oc_ata_ocidec1	693	660	692	1.5	1.6	4.48
oc_ata_ocidec2	838	769	837	1.55	1.72	4.77
oc_ata_v	512	452	494	0.15	0.2	3.10
oc_cordic_p2r	3175	3131	3175	0.98	19.72	9.12
oc_correlator	611	585	610	0.13	0.22	9.35
oc_dct_slow	513	479	512	2.87	36.7	6.50
oc_des_area_opt	1192	1127	1191	6.63	31.15	6.52
oc_des_des3area	1784	1580	1783	3.77	29.32	5.35
oc_i2c	597	548	590	3.25	2.45	28.67
oc_rtc	887	708	881	0.43	0.65	22.13
oc_sdram	731	648	728	0.32	1.5	4.08
os_sdram16	947	821	923	0.45	2.4	69.75
geomean	980	877	970	0.93	3.07	9.42
normalized mean	1	0.9	0.99	1	3.29	10.08

Table 3.2: Comparisons of area and runtime between ABC and ROSE using robust and area-efficient PLB templates

	MTBF (year)	
benchmark area	ABC	ROSE using R-PLB
330,000 LUTs	20.66	27.15
ratio	1	1.31

Table 3.3: MTBF (mean time between failures)

CHAPTER 4

IPR: In-Place Reconfiguration for FPGA Reliability

To ensure not breaking the current CAD flow during the reliability optimization, we present a logic synthesis algorithm which performs a satisfiability (SAT) based in-place reconfiguration (IPR) in look-up tables (LUTs) on FPGAs. IPR maximizes identical configuration bits for complementary inputs of a LUT to prevent the error propagation. It preserves the functionality and topology of the netlist, and therefore, requires no change of physical design. Compared to the state-of-the-art academic technology mapper Berkeley ABC, IPR reduces the relative fault rate by 48% and increases mean time to failure (MTTF) by 1.94× with the same area and performance. Applying both ROSE and IPR reduces the relative fault rate by 49% and increases MTTF by 2.40× with 19% less area.

4.1 Introduction

Compared to application specific integrated circuits (ASICs), field programmable gate arrays (FPGAs) are more vulnerable to soft errors, considering that most of logic functions and interconnects are implemented by SRAM cells. FPGAs used in applications, such as networking, power grid, medical equipment, automotive system, are all suffering from the reliability issues. Soft error mitigation should be considered during the design flow in FPGA applications. In this chapter, we focus on logic synthesis level to enhance the reliability of FPGA based designs.

Recently, a robust logic resynthesis technique called ROSE has been proposed [HFH08] as an effective design optimization for fault tolerance. The technique rewrites a LUT-based Boolean network and inserts logic masking to prevent the propagation of stochastic faults. It obtains 2× MTTF improvement with no area and performance overhead compared to the state-of-the-art academic logic synthesis tool Berkeley ABC [Mis11]. ROSE is orthogonal to the existing fault tolerance techniques such as [LV62, LH07, CLH08, PCM07, KPM07], with advantages of negligible overhead on area, performance, and testing. However, ROSE can change the topology of the LUT-based logic network, which limits its applicability in the design flow as shown below.

In the design flow of FPGA based systems (see Figure 4.1), logic resynthesis (e.g., ROSE) is performed after logic synthesis and before physical design. There is a drawback for most existing techniques [LV62, LH07, CLH08, PCM07, KPM07, HFH08] for FPGA reliability in the aforementioned flow. They rely on the results of placement and routing to gather the fault information, but they change interconnects, i.e., change the physical design. Due to the change of the physical design during the optimization, the fault information is changed and needs to be updated, which requires a redo of placement and routing and physical aware logic optimization, resulting in the reliability convergence issue.

Therefore, there is a need of in-place logic optimization for the reliability. In this chapter, we propose an *in-place* logic resynthesis algorithm, which performs logic transformation while preserving the functionality and the topology of the LUT-based logic network. Therefore, it does not require a redoing of the physical design and leads to a faster design closure. Our core algorithm *in-place reconfiguration* (IPR) maximizes identical configuration bits corresponding to complementary inputs of a LUT such that the faults seen at a pair of complementary inputs have less possibility of propagation and the overall reliability is optimized. IPR is iteratively carried out by simultaneously reconfiguring multiple adjacent LUTs

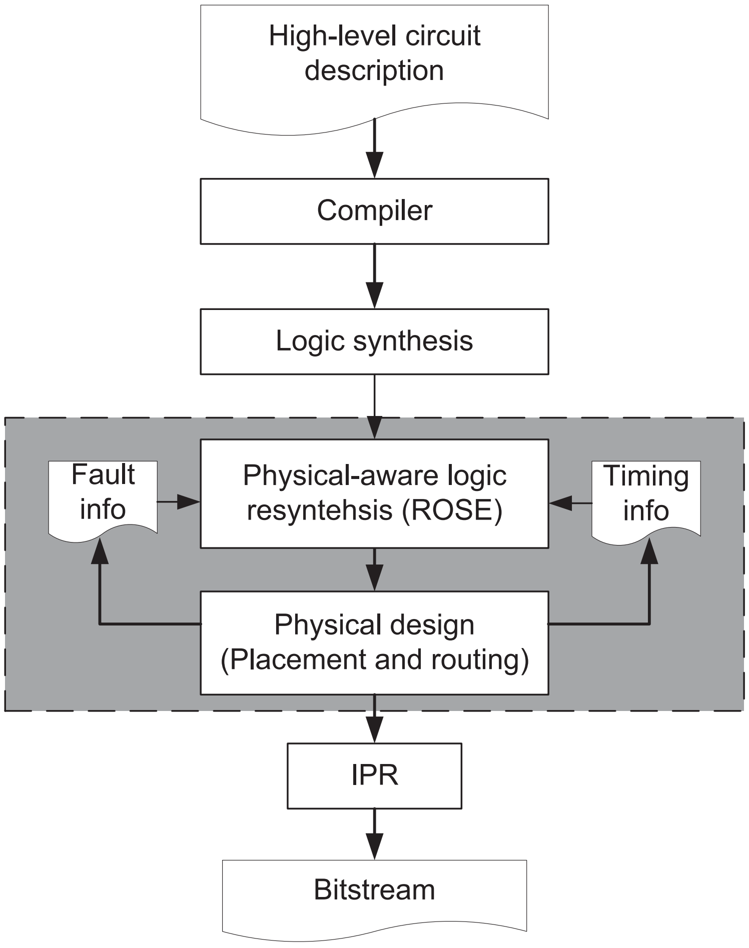

High-level circuit description
Compiler
Logic synthesis
Fault info
Physical-aware logic resyntehsis (ROSE)
Timing info
Physical design (Placement and routing)
IPR
Bitstream

Figure 4.1: A simplified synthesis flow for FPGAs

without changing the functionality and topology of the LUT-based logic network. IPR can apply to both combinational and sequential circuits.

Compared to the state-of-the-art academic technology mapper Berkeley ABC, IPR reduces the relative fault rate by 48% and increases MTTF by 1.94×. IPR is orthogonal to ROSE since we can first perform ROSE to increase the robustness before the physical design, and then employ IPR as a post physical design optimization for further reliability enhancement. Combining ROSE and IPR reduces the relative fault rate by 49% and increases MTTF by 2.40X compared with Berkeley ABC mapper.

The remainder of this chapter is organized as follows: Section 4.2 presents background and preliminaries. Section 4.3 and Section 4.4 provide the in-place LUT reconfiguration algorithm and the proposed logic resynthesis algorithm, respectively. The experimental results are given in Section 4.5 and the chapter is concluded with future research directions in Section 4.6.

4.2 Preliminaries

4.2.1 In-place Resynthesis

In-place resynthesis is a technique that optimizes a circuit while preserving the functionality and topology of the logic network. The major advantage of this technique is that it has no impact on the design closure because the optimization is performed in place. In-place resynthesis algorithms are particularly feasible for various FPGA optimizations due to the design freedom provided by LUT configurations. For example, a LUT truth table reset technique is used for dynamic power reduction considering the floating pins of LUTs [GA07], and a polarity selection was proposed for leakage power reduction based on the dependency of leakage power on logic values of signals [AN06]. The aforementioned work reconfigures individual LUTs. In contrast, the in-place LUT reconfiguration approach proposed in this chapter reconfigures multiple LUTs simultaneously.

4.2.2 Circuit Representation

A *K-LUT* is a LUT with K inputs, one output, and 2^K LUT configuration bits. A LUT-based *Boolean network* is represented using a directed acyclic graph (DAG), in which nodes denote LUTs and directed edges represent the interconnects among the LUTs. The nodes in the lowest level of the DAG are called *circuit inputs* (CIs), which include the *primary inputs* (PIs) and the outputs of registers. The nodes in the highest level are called *circuit outputs* (COs), which include *primary outputs* (POs) and the inputs to registers.

A *fanin* (resp. *fanout*) *cone* of node n is a sub-network whose nodes can reach the fanin edges of n (resp. can be reached from the fanout edges of n). A *maximum fanin* (resp. *fanout*) *cone* of node n is the largest fanin (resp. fanout) cone of n. It contains all the nodes in the path from CI (resp. CO) to node n. A *fanin cut* (resp. fanout cut) C of node n is a set of nodes in the network such that each path from a CI (resp. CO) to n passes through at least one node in C, and node n is called the *root* of *cut* C.

4.2.3 Fault Model

We assume a stochastic single fault model, i.e., at a time, at most one fault occurs. The faults have identical random distribution for configuration bits in LUTs. The *fault rate* of a circuit is the percentage of primary input vectors that cause observable primary output errors. Although we assume single fault in our experiments, our algorithm (described in Section 4.3 and 4.4) can handle multiple faults.

4.2.4 ODC Mask and Node Criticality

In this chapter, we utilize don't cares for the fault tolerance. There are two kinds of don't cares. The *satisfiability don't cares* (SDCs) are due to some combinations not being produced as input vectors of nodes. The *observability don't cares* (ODCs) occur, when output values of nodes do not propagate to the COs under some conditions [MB05b].

We adopt the definition of ODC mask [KPM09] to quantify the capability of fault tolerance of the node under tested during the optimization. The ODC mask of node n is defined by Definition 2. Given the definition, we define the *criticality* of node n as the percentage of ones in its ODC mask, which is used to decide the ordering of nodes for in-place reconfiguration.

Definition 2. ODC Mask *Let $\langle X_1, \ldots, X_K \rangle$ be a sequence of K input vectors for node n. The* ODC mask *of n, written **ODCmask**(n), is a K-bit sequence where ith bit is 0 if the input vector X_i is in the don't care set of n; otherwise, the ith bit is 1. Formally, **ODCmask**$(n) \in 0, 1^K$ such that **ODCmask**$(n)_i \equiv X_i \notin$ **ODC**(n), where $ODC(n)$ is the don't care set of n.*

The ODC mask is computed as follows. First, the truth table of node n is bitwise negated. Then, a simulation is performed in the fanout cone to check whether the changes can propagate to primary outputs. Plaza et al. proposed a heuristic algorithm [PCM07] with $O(N)$ complexity to calculate the ODC mask for all the nodes. In reverse topological order, for each node, a local ODC mask is simulated for its immediate fanout. The local ODC mask is computed by negating the truth table of n and checking whether the output of the fanout has been changed. Then, the local ODC mask is bitwise-ANDed with the global ODC mask to produce the branch ODC mask for that fanout. After that, all the branch ODC masks for fanouts are bitwise-ORed to produce the global ODC mask for node n.

4.3 IPR Algorithm

According to criticalities of nodes, we apply in-place reconfiguration for nodes sequentially to increase the robustness of the circuit. The key idea of IPR is to maximize identical configuration bits in a selected logic block, while preserving the functionality and the topology of the block by reconfiguring multiple LUTs simultaneously. In each reconfiguration, in-place Boolean matching is used to ensure the functional equivalence. In this way, transient or permanent faults seen at a pair of complementary inputs have less possibility of propagation, and the overall reliability of the circuit is optimized. This procedure can be applied to a sequential logic network by applying independently to each combination logic block.

In the rest of the chapter, without specific declaration, we denote the node under optimization as n_{opt}. For n_{opt}, an input vector is corresponding to a logic output specified by a configuration bit, e.g., input vector 01 generates logic output 0 if the configuration bit c_{01} is 0. The input vector and the configuration bit have a one-to-one relationship. Furthermore, the configuration pair (c_{00}, c_{10}) is corresponding to a pair of complementary inputs 00 and 10 with regards to input pin 1.

4.3.1 Motivation

In this section, we show an example to illustrate how to enhance the robustness of the circuit by maximizing identical configuration bits for complementary inputs of a LUT. In figure 4.2, there are two 2-LUTs. Suppose input 1 has a high defect rate. Given the same input sequences, there are two different output sequences for the two 2-LUTs. When $0 \to 1$ faults occur in input 1, the fault rate of LUT A is greater than that of LUT B. The reason is that in LUT B, the value of configuration bits c_{00} and c_{10} are same that introduces logic masking for input 1 when the value of input 0 is 0. Intuitively, the more identical pairs of configuration bits for complementary inputs are in the LUT, the more faults are prevented from propagation. However, reconfiguring can change the function of a LUT. Reconfigure multiple LUTs simultaneously in a selected logic block can be used to ensure the functionality of the block unchanged.

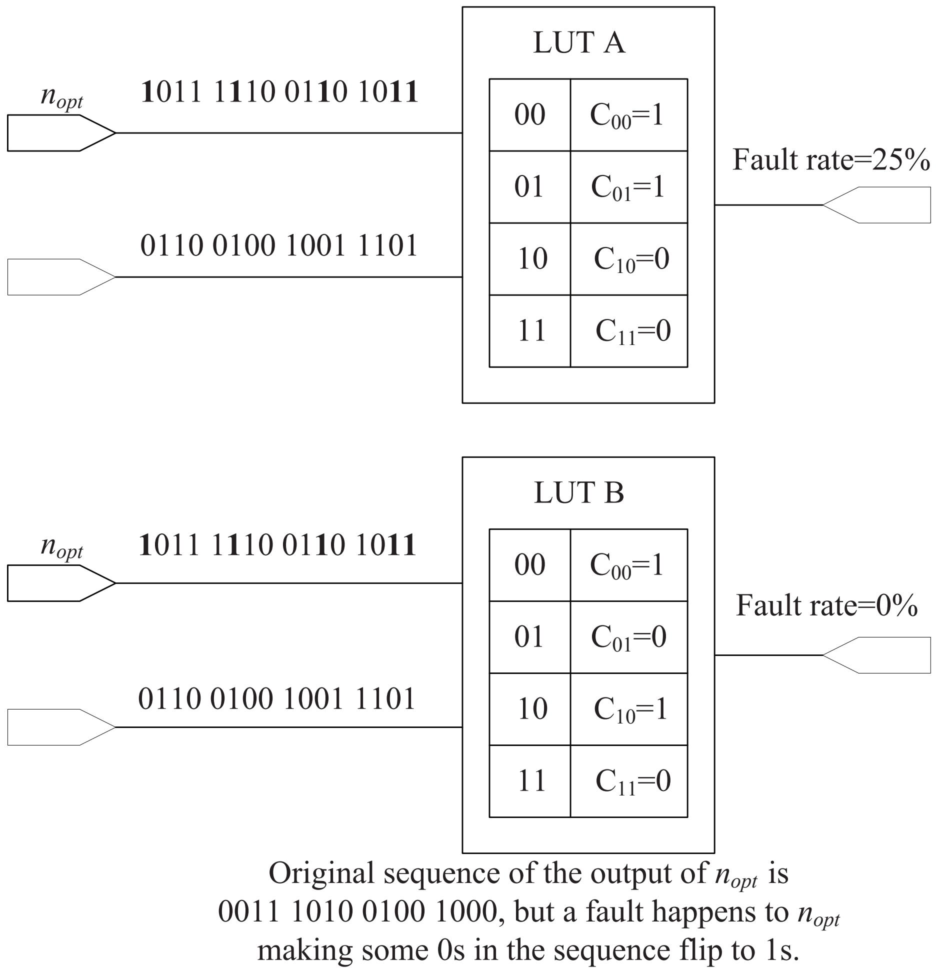


Figure 4.2: Motivation of the in-place reconfiguration algorithm

4.3.2 Algorithm Overview

The overview of the IPR algorithm is illustrated in Algorithm 1. Given a node n_{opt}, we denote all the fanout cuts of the node as $\mathcal{S}_{\mathcal{C}}$. We represent all the pairs of configuration bits for complementary inputs in each fanout cut as $\mathcal{S}_{\mathcal{P}}$. First, we calculate the criticalities of all the configuration bits for node n_{opt}. After that, each fanout cut c_i is processed sequentially according to the size of the cut. In each iteration, $\mathcal{S}_{\mathcal{P}}$ is initialized as all the pairs of configuration bits for complementary inputs in current fanout cut. Then, we iteratively search a feasible cone containing c_i by function **constructCone** (c_i). We check whether there is a LUT

Algorithm 1 IPR(n_{opt})

```
1: Calculate the criticalities of all the configuration bits for node $n_{opt}$;
2: $Valid \leftarrow$ false;
3: // For each fanout cut $c_i \in \mathcal{S_C}$ according to the size of the cut;
4: repeat
5:     $\mathcal{S_P} \leftarrow$ All the pairs of configuration bits of the cut $c_i$;
6:     while $\mathcal{S_P} \neq \Phi$ do
7:         $Cone(c_i) \leftarrow$ constructCone ($c_i$);
8:         $Valid \leftarrow$ booleanMatching ($\mathcal{S_P}, Cone(c_i)$);
9:         if $Valid$ then
10:            inPlaceReconfiguration ($\mathcal{S_P}, Cone(c_i)$);
11:            break;
12:        else
13:            $\mathcal{S_P}$ = $\mathcal{S_P}$ - $p_i$;
14:        end if
15:    end while
16: until $\mid \mathcal{S_C} \mid$ == 1 || $Valid$ == true
```

configuration for all LUTs in the cone that ensures all the pairs in $\mathcal{S_P}$ have identical values without changing the function and topology of the cone by function **booleanMatching** ($\mathcal{S_P}, Cone(c_i)$). If there is a solution, the configuration is applied by **inPlaceReconfiguration** ($\mathcal{S_P}, Cone(c_i)$). Otherwise, the pair of configuration bits with the least criticality is removed from $\mathcal{S_P}$, and a new round is invoked with the updated $\mathcal{S_P}$. Finally, we terminate the IPR algorithm when the size of $\mathcal{S_C}$ is 1 or we find a feasible solution by the in-place Boolean matching.

In the following sections, we discuss items including (i) the calculation of the criticality for the configuration bit; (ii) the cone construction strategy used in **constructCone** (c_i); and (iii) in-place Boolean matching for LUT reconfiguration preserving the function and topology of the selected cone.

4.3.3 Criticality for Configuration Bit

Given the complementary inputs of a LUT, we cannot set all the pairs of configuration bits identical. Choosing which pairs of configuration bits to be identical is the key for the in-place optimization. Therefore, we have to define the criticality of configuration bit. High priority should be given to the configuration bit which can mask more faults during in-place reconfiguration. In Equation 4.1, $N_{sequence}$ denotes the length of the sequence of input vectors used for the full-chip functional simulation. N_{vector} is the number of input vectors associated with the configuration bit c in the sequence. $R_{tolerate}$ is the percentage of input vectors, which masks faults when the input of the LUT is defected. $R_{tolerate}$ is derived from ODC mask. 1-$R_{tolerate}$ represents the percentage of input vectors making the faults observable at primary outputs. The criticality of configuration bit c can be formulated as follows,

$$Criticality\_bit(c) = \frac{N_{vector}}{N_{sequence}}(1 - R_{tolerate}) \tag{4.1}$$

4.3.4 Cone Construction

In the IPR algorithm, when a node n_{opt} is selected, a cone surrounding this node needs to be chosen for reconfiguration. We choose the cones with following two criteria. Given a node n_{opt}, the cone needs to cover nodes in the fanout as many as possible; besides, the input size of the cone is limited. The first criteria tries to ensure that we could find a feasible solution during Boolean matching, given the flexibility provided by the node coverage; the second criteria guarantees computation cost for Boolean matching not be too high.

4.3.5 In-place Boolean Matching

The difference between in-place Boolean matching(IP-BM) and other template-based Boolean matching is that, we use the original cone in the netlist instead of the pre-designed template to rewrite the netlist. We employ IP-BM to check whether we can reconfigure LUTs within the cone and make selected pairs of configuration bits (specified by set $\mathcal{S}_{\mathcal{P}}$ in Algorithm 1) identical without changing the function and the topology of the cone. If such a configuration exists, IP-BM should return a set of configurations for all LUTs within the cone. Otherwise, a configuration of the cone with new specified pairs of configuration bits is used to invoke a new round of IP-BM.

IP-BM is formulated as follows. Suppose the cone C_F has m inputs, i.e., $x_1, \cdots, x_m$ and one output, F. The truth table of output F is denoted by $(F_0, F_1, \cdots, F_{2^m-1}), F_i \in \{0,1\}$, which is computed by simulation. To encode IP-BM as an SAT problem, we first encode this truth table into a conjunctive normal form (CNF) as follows.

$$\begin{aligned}\Psi(F) \;=\;& (\overline{x_1} \wedge \overline{x_2} \wedge \cdots \wedge \overline{x_m} \rightarrow F_0) \wedge \\ & (x_1 \wedge \overline{x_2} \wedge \cdots \wedge \overline{x_m} \rightarrow F_1) \wedge \cdots \wedge \\ & (x_1 \wedge x_2 \wedge \cdots \wedge x_m \rightarrow F_{2^m-1}), \end{aligned} \tag{4.2}$$

where $F_i = F$ if $F(i) = 1$, and $F_i = \overline{F}$, otherwise.

To encode the cone, we formulate inputs and outputs of cone C_F as variables in the CNF. For example, a K-input LUT (with inputs $x_0^L, \cdots, x_K^L$, output z and configuration bits $c_0, \cdots, c_{2^K-1}$) can be encoded as

$$\begin{aligned}\Psi(\mathcal{L}) =\;& (\overline{x_0^L} \wedge \overline{x_1^L} \wedge \cdots \wedge \overline{x_K^L} \rightarrow (z \leftrightarrow c_0)) \wedge \cdots \wedge \\ & (x_0^L \wedge x_1^L \cdots \wedge x_K^L \rightarrow (z \leftrightarrow c_{2^K-1}))\end{aligned}$$

Suppose cone C_F contains p LUTs: $L_0, \cdots, L_{p-1}$. The characteristic function of this cone is obtained by taking the conjunction of constraints defining these LUTs as follows.

$$\Psi(C_F) = \Psi(L_0) \wedge \cdots \wedge \Psi(L_{p-1}). \tag{4.3}$$

To make the values of a pair of configuration bits (c_i, c_j) identical, we have the following constraint clause:

$$c_i \leftrightarrow c_j \tag{4.4}$$

Combining (4.2), (4.3) and (4.4), we form the following CNF formulation for IP-BM:

$$\begin{aligned}&\exists c_1 \cdots c_n \forall x_1 \cdots x_m \exists z_1 \cdots z_p \,.\, \Psi(C_F) \wedge \Psi(F) \wedge \\ &(G \leftrightarrow F) \textstyle\bigwedge_{\forall (c_i,c_j)\in \mathcal{S}_{\mathcal{P}}} (c_i \leftrightarrow c_j)\end{aligned} \tag{4.5}$$

where G is the output of the cone. Note that (4.5) is a quantified Boolean formula (QBF) and it can be solved by any QBF solvers or SAT solvers after eliminating the universal quantifiers [LSB05b, CM07, HSM07]. If (4.5) is satisfiable, there is a feasible configuration of the cone such that all pairs of configuration bits defined in set $\mathcal{S}_{\mathcal{P}}$ can be set identical, and the configuration of LUTs can be obtained based on the assignments in $c_1, \cdots, c_n$. The topology of the cone is not changed after reconfiguration considering that it is constrained by characteristic function (4.3). If (4.5) is unsatisfiable, we need to reduce the size of set $\mathcal{S}_{\mathcal{P}}$ and solve the IP-BM with less constraints as shown in Algorithm 1.

4.4 Robust Logic Resynthesis

4.4.1 Overall Algorithm

We integrate the IPR algorithm into the logic resynthesis flow. First, the circuit analysis is performed. We simulate the entire circuit to gather the truth table for each node in the topological order and calculate the ODC mask [PCM07] in the reverse topological order. The criticality of each node is calculated based on the ODC mask. According to the descendant order of criticalities of nodes, we iteratively select a node n_{opt} to be optimized by the IPR algorithm until time is out. After we reconfigure fanouts of n_{opt}, we update the truth table and ODC mask of the reconfigured cone locally. The node criticality is re-calculated before moving to the next node. Figure 4.3 outlines the flow of robust logic resynthesis.

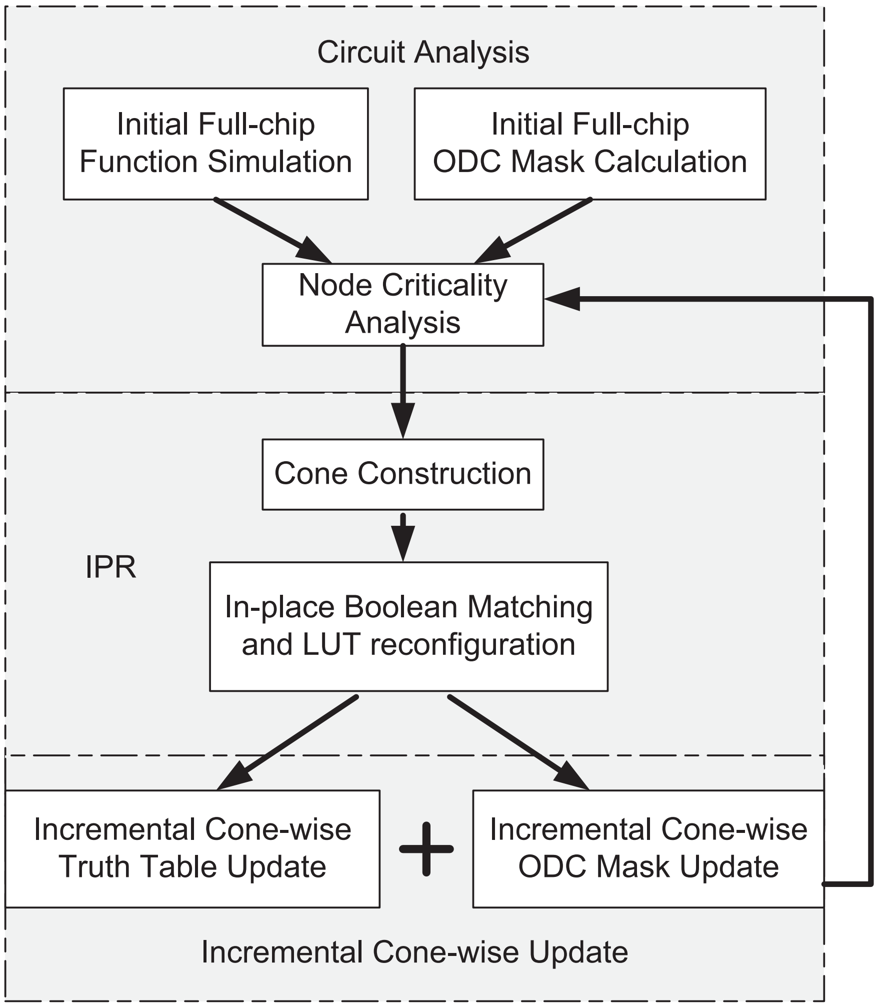

Circuit Analysis
Initial Full-chip Function Simulation
Initial Full-chip ODC Mask Calculation
Node Criticality Analysis
Cone Construction
IPR
In-place Boolean Matching and LUT reconfiguration
Incremental Cone-wise Truth Table Update
+
Incremental Cone-wise ODC Mask Update
Incremental Cone-wise Update

Figure 4.3: The flow of robust logic resynthesis

4.4.2 Localize Update

We denote the optimized cone as C_R. The maximum fanin cone and maximum fanout cone of C_R are denoted as C_{MFI} and C_{MFO}, respectively. After performing the IPR algorithm, there are two kinds of node information to be updated. One is the truth table, and the other is the ODC mask.

For the truth table, because we preserve the function of the circuit outside cone C_R during Boolean matching, only the truth tables of LUTs in cone C_R are changed and need to be updated.

The ODC mask of C_R has been changed by IPR. Because C_R is in the maximum fanout of C_{MFI} and the ODC mask calculation is based on the maximum fanout cone of a node, the ODC mask of C_{MFI} is also changed. C_{MFO} is unaffected during the optimization, therefore, its ODC mask is not changed. It is time-consuming to update the ODC mask for C_{MFI}. Fortunately, our experiments show that the reliability improvement of the circuit is not affected, even if we do not update the ODC mask of C_{MFI}. In our algorithm, we only update the ODC mask of C_R.

4.5 Experimental Results

We have implemented the IPR algorithm in C++ and used miniSAT2.0 [ES] as the SAT solver. All experimental results are collected on a Ubuntu workstation with 2.6GHz Xeon CPU and 2GB memory. We test our algorithms on QUIP benchmarks [ALT]. As discussed in Section 4.2.3, we assume that all configuration bits have an equal possibility to be defective. We calculate the fault rate by Monte Carlo simulation with 20K iterations where one bit fault is randomly injected in each iteration for 1k input vectors.

As shown in Figure 4.4, we first map each benchmark by the Berkeley ABC mapper [Mis11] using 4-LUTs, then we perform and compare the following synthesis flows: (1) ABC followed by the physical design tool, VPR [BR97], without any defect-oriented logic resynthesis, (2) ABC followed by VPR, and in-place optimization by IPR, and (3) ABC followed by ROSE and VPR, and in-place optimization by IPR. Considering faults in configuration bits of both LUTs and interconnects, Monte Carlo simulation is performed to calculate the full-chip fault rate, with results summarized in Table 4.1.

We use ABC as the baseline for comparison. As shown in Table 4.1, IPR reduces the fault rate by 48% without area overhead. The best flow in terms of the robustness and area is ROSE+IPR, which reduces fault rate by 49% with 19%

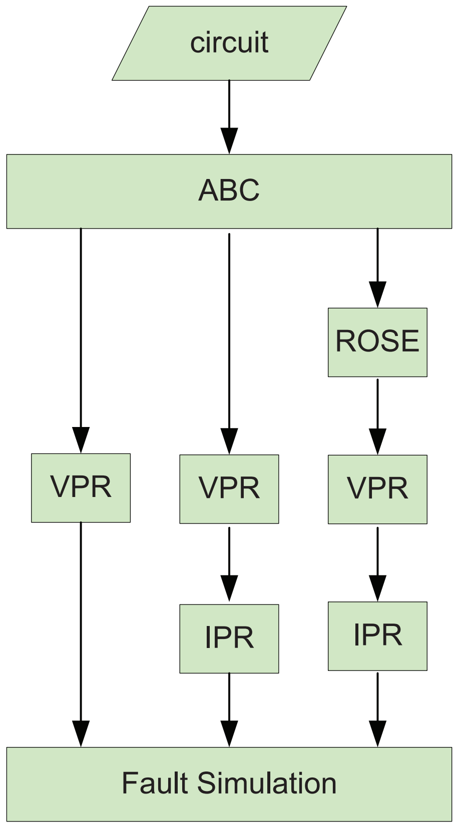


Figure 4.4: Experimental flows

smaller area. However, for some circuits, ROSE+IPR flow may result in higher fault rate than IPR flow (e.g., barrel64 and fip_cordic_cla). That is because ROSE is performed before IPR, which changes the topology of the LUT-based netlist and changes the input netlist for IPR. Table 4.1 also reports MTTF. Because we assume that all the testing conditions are the same for ABC and IPR, MTTF is inversely proportional to the product of fault rate and area. Compared to ABC, IPR and ROSE+IPR increases MTTF by 1.94$\times$ and 2.40$\times$, respectively.

QUIP Benchmark							
	Fault Rate			LUT#			Runtime (seconds)
	ABC	IPR	ROSE+IPR	ABC	IPR	ROSE+IPR	IPR
barrel64	2.02%	0.93%	0.94%	1931	1931	1484	19.46
fip_cordic_cla	1.92%	0.78%	0.79%	1042	1042	802	6.11
fip_cordic_rca	1.91%	0.83%	0.77%	981	981	751	5.71
mux8_128bit	5.37%	2.84%	2.74%	1923	1923	1796	1.52
oc_ata_ocidec1	2.98%	1.83%	1.80%	695	695	632	4.52
oc_ata_ocidec2	3.20%	1.69%	1.73%	840	840	742	4.89
oc_ata_v	2.15%	1.13%	1.11%	514	514	411	1.43
oc_dct_slow	1.55%	0.90%	0.85%	509	509	439	4.91
oc_des_area_opt	1.59%	0.98%	0.99%	1190	1190	1007	20.51
oc_des_des3area	1.68%	1.16%	1.23%	1782	1782	1479	39.64
oc_rtc	1.59%	0.72%	0.77%	879	879	591	3.90
oc_sdram	1.97%	0.89%	0.86%	729	729	553	1.89
os_sdram16	1.65%	0.79%	0.80%	947	947	719	5.28
GeoMean	2.12%	1.09%	1.09%	977.72	977.72	791.10	5.58
Ratio	1	0.52	0.51	1	1	0.81	
MTTF	1	1.94	2.40				

Table 4.1: Comparison of fault tolerance among ABC, IPR, and IPR + ROSE flow

4.6 Conclusions and Future Work

IPR increases MTTF by $2\times$ over ABC, the state-of-the-art academic technology mapper. More importantly, unlike its predecessors, IPR preserves the topology of the logic network for a faster design closure between logic synthesis and physical design. In addition, IPR is complementary to existing fault-tolerant resynthesis algorithms. Combining IPR and ROSE, the best design flow for reliability and design closure is to perform pre-layout ROSE and post-layout IPR.

Although our experiments assume single fault, the proposed algorithm can deal with multiple uncorrelated faults, which will be explored in future work. We will also extend the proposed algorithm to consider given correlations between faults.

While our criticality calculation in this chapter does not consider interconnect explicitly, IPR algorithm can be used without change if interconnects are taken into consideration for criticality. In addition, the current algorithm tolerates interconnect defects implicitly by modeling them as defects on outputs of a LUT. In the future, we will extend IPR with criticality considering interconnects explicitly.

CHAPTER 5

IPF: In-Place X-Filling Algorithms for the Reliability of Modern FPGAs

Targeting the modern FPGA architecture including both LUTs and interconnects, we propose three synthesis based in-place X-Filling algorithms by utilizing don't cares to augment the reliability of FPGA based designs. Compared with circuit and architecture based solutions, our algorithms are in place, and do not require area, power, performance, and design time overheads. Compared with other synthesis based algorithms, we take into account the widely accepted interconnect architecture besides considering LUTs during optimization. For the ten largest combinational MCNC benchmark circuits mapped to 6-LUTs, our approaches achieve up to a 37% greater failure rate reduction, and up to 150× runtime speedup, compared with the best known in-place algorithm, the In-Place Decomposition (IPD) algorithm.

5.1 Introduction

Field Programmable Gate Arrays (FPGAs) have been increasingly used in many applications due to fast time-to-market, no Nonrecurring Engineering Expense (NRE), and easy long-term maintenance. They have been widely used in different applications, such as networking, digital signal processing, prototyping and hardware emulation. Nevertheless, Single Event Upsets (SEUs), also called soft errors, have posed a major barrier for the reliability of SRAM-based FPGAs.

SEUs are generally caused by high-energy particle strikes, e.g., neutrons coming from cosmic rays or alpha particles emitting from trace impurities in packaging materials and solder bumps [Bid10]. Other reasons cause SEUs include the power supply disturbance and the electromagnetic interference. They change values of devices, such as SRAM cells and flip-flops, when charges collected from strikes are larger than a threshold. Because most logic functions and interconnects in SRAM-based FPGAs are implemented by SRAM cells, they are more vulnerable to SEUs compared with Application Specific Integrated Circuits (ASICs). SEUs have a permanent impact on FPGAs till configuration scrubbings are applied [SMC01]. In the past, the SEU issue has received attention only from high reliability applications in military and aerospace areas. As modern FPGAs have advanced to 28nm technology to provide low cost and low power solutions [Meh12, ALT12], the devices are prone to SEUs for most applications due to reduction in core voltage, decrease in transistor geometry, and increase in switching speed. Therefore, SEU mitigation for SRAM-based FPGAs has gained growing significance.

The chapter focuses on SEU mitigation for SRAM-based FPGAs. There are a number of studies in the literature on the subject. The solutions can broadly be divided into circuit, architecture, and synthesis based techniques. The first two categories require extensive area, power, performance, design time, or cost overhead [XIL12b, LV62, SRK04, Tam06, MSZ05]. Several studies have demonstrated that the SEU issue can be mitigated by synthesis based approaches while minimizing the aforementioned overheads. However, they either mitigate errors introduced by SEUs on Look-Up Tables (LUTs) only [HFH08,FHH09,LFH10,CM10], without considering the SEU impact on interconnects, or there is a flaw in the interconnect SEU model, e.g., the model in Jose's paper [JHM10] assumed that there is only one configuration bit in each net. They rely on creating don't cares to tolerate errors introduced by SEUs. However, the large amount of pre-existing don't cares makes them difficult to further increase don't cares. As shown in Table 5.1, for the ten largest combinational MCNC benchmark circuits [Yan91], we observe that don't cares comprise approximately 40% and 60% of utilized LUT configuration bits, when the designs are mapped to 4-LUTs and 6-LUTs [1], respectively.

This motivates us to exploit pre-existing don't cares in LUTs to augment the reliability of designs. We present three In-Place X-Filling (IPF) algorithms [2], which fill don't cares to mask errors introduced by SEUs on both LUTs and interconnects. Compared with other synthesis based algorithms, we take into account the widely accepted interconnect architecture used in VPR [BR97, LKJ11] during optimization. In addition, our algorithms overcome the slow runtime issue prevailing in most previous synthesis based techniques. That is because our algorithms do not search for functionally equivalent implementations, therefore, they do not need time-consuming algorithms like Binary Decision Diagrams, Boolean Satisfiability [HFH08, FHH09], Integer Linear Programming [LFH10], or Set of Pairs of Functions to be Distinguished [JHM10]. Compared with circuit and architecture based solutions, our algorithms do not require area, power, performance, and design time overheads, because they do not change LUT level placement and routing, i.e., they are in-place algorithms.

For the ten largest combinational MCNC benchmark circuits mapped to 6-LUTs, our approaches achieve up to a 37% greater failure rate reduction, and up to 150× runtime speedup, compared with the best known synthesis based in-place algorithm, the In-Place Decomposition (IPD) algorithm. In this chapter, the sensitivity of a circuit to SEUs is measured by the failure rate. The failure rate is the frequency with which a circuit fails due to SEUs occur on its LUT configuration bits or interconnect configuration bits. The computation of the failure rate is presented in 5.2.2.

[1]After designs are mapped by the Berkeley mapper [Mis11], their don't cares are computed by the windowing technique proposed by Cong et al. [CM10].

[2]The term has been used for power-aware Automatic Test Pattern Generation (ATPG) [LHM09], in which power is minimized by filling don't cares to reduce logic switches of designs.

Table 5.1: The ratio of don't cares to utilized LUT configuration bits. *SDC refers to satisfiability don't care.

MCNC circuits	4-LUT			6-LUT		
	Configuration bits#	Don't cares#	SDC* bits#	Configuration bits#	Don't cares#	SDC bits#
alu4	11520	3662	3151	33408	17180	15356
apex2	15440	7785	3570	45760	29063	19303
apex4	12640	4685	4622	37440	20920	20319
des	19984	6003	5597	54528	33185	31686
ex1010	17648	5800	5746	43584	23445	23105
ex5p	8656	4646	4331	23808	16193	13471
misex3	11776	3892	3552	31552	15923	14751
pdc	35360	18564	17412	105280	63760	58580
seq	15968	5342	4187	46208	20885	17934
spla	34000	17726	16625	98688	61325	56294
Average	18299	7811	6879	52026	30188	27080
Ratio	1	**42.68%**	37.59%	1	**58.03%**	52.05%
		1	**88.08%**		1	**89.70%**

The rest of this chapter is organized as follows. We start with preliminaries introducing design representation, and don't cares in Section 5.2. The IPF problem is formulated in Section 5.3, and the proposed algorithms are presented in Section 5.4. The experimental results are summarized in Section 5.5 followed by a conclusion in Section 5.6.

5.2 Preliminaries

Before introducing our algorithms, we first present the FPGA architecture that our algorithms target. We also show the representation of a FPGA design used in following sections. In addition, we introduce the fundamentals of don't cares.

5.2.1 Design Representation

Before discussing about the background, basic notations about design representation are provided as follows. An FPGA design is usually represented by a directed acyclic graph. In the graph, nodes represent LUTs, and edges represent interconnects between LUTs. If Node a drives Node b, Node a is called Node b's $fan-in$, and Node b is called Node a's $fan-out$. The node without fan-in is called *primary input*, and the node without fan-out is called *primary output*. The $fan-in$ ($fan-out$) *cone* of Node a is the nodes reachable through fan-in (fan-out) edges from Node a.

5.2.2 Failure Rate and Don't Care

In this chapter, the sensitivity of a configuration bit to an SEU is measured by the failure rate of the configuration bit C_i, i.e., the frequency with which a circuit fails due to the SEU on the configuration bit, expressed in Equation 5.1. The sensitivity of a chip to SEUs can be measured by the failure rate of the whole chip, which is the average of failure rates of all the configuration bits (See Equation 5.2). V denotes all the combinations of primary input vectors. C denotes all the configuration bits. PO_{golden} is the primary output vector without the impact of SEUs. PO_{SEU} is the primary output vector when an SEU occurs on the configuration bit C_i.

$$Fr(C_i) = \frac{\sum_{\forall v \in V}(PO_{golden}(v) \wedge PO_{SEU}(v)(C_i))}{|V|} \tag{5.1}$$

$$Fr = \frac{\sum_{\forall C_i \in C} Fr(C_i)}{|C|} \tag{5.2}$$

If a configuration bit is insensitive to an SEU, i.e., flipping the bit doesn't change the functionality of the logic network, the bit is a don't care bit. There are two kinds of don't care bits, i.e., satisfiability don't care (SDC) bits and observability don't care bits. As a result of reconvergent paths and fan-out cone masking, there is limited accessibility and observability to some nodes, specifically, some configuration bits insides those nodes. The SDC bit is the inaccessible configuration bit inside the nodes which don't have a full set of permutations at their fan-ins [MZS06]. The ODC bit is the configuration bit which is not observable at the primary output given a set of primary input vector [MZS06]. In Figure 5.1, C_{11}[3] in LUT D is an SDC bit that is not accessible. C_{00} in LUT A is an ODC bit when $a = 0$ and $d = 0$. In this work, we only focus on exploiting SDC bits for SEU mitigation for two reasons: (1) SDC bits comprise about 90% of total don't cares for the designs under test, as shown in Table 5.1; (2) SDC bits are compatible don't cares, because flipping an SDC bit does not invalidate other don't cares.

5.3 Problem Formulation

In this section, we illustrate utilizing pre-existing don't cares to mitigate errors introduced by SEUs, and formulate the In-Place X-Filling (IPF) problem. In Figure 5.1, given a logic function f, there are two implementations with same interconnects between LUTs. Configuration bit C_{11} in LUT D is an SDC bit that is inaccessible in a normal situation. Therefore, the functionalities of two implementations are the same regardless of whether C_{11} is assigned 0 or 1. Nevertheless, when C_{11} is filled with 0 in Figure 5.1 (a), the failure rate is greater than when it is assigned 1 in Figure 5.1 (b). Both failure rates in Figure5.1 (a)(b) are calculated by Equation 5.1 and5.2. The reason is that, SDC bit C_{11} in LUT D may be accessed when an SEU is in the fan-in cone of LUT D; therefore, when C_{11} is filled with a feasible value, even if an SEU occurs, C_{11} can be used to mask errors in LUT D. The example shows that, we can exploit pre-existing don't cares instead of creating don't cares to augment the reliability of designs while preserving the functionality and the topology of designs.

The basic idea hidden behind the example is that, in a normal circuit, SDC bits in LUTs are inaccessible. When SEUs occur in fan-in cones, SDC bits might

[3]In the chapter, the configuration bit corresponding to the input ABCD is denoted as C_{ABCD}.

be chosen. In this situation, LUTs can still output correct values, if SDC bits are feasibly assigned. More concretely, we formulate the in-place X-Filling problem in Definition 3.

Definition 3. ***The IPF problem:*** *Given a design, fill SDC bits in all LUTs to increase the likelihood of masking errors introduced by SEUs in their fan-in cones thereby to augment the reliability of the design.*

5.4 In-place X-Filling Algorithms

In this section, three IPF algorithms are proposed. The three algorithms are in two categories. Two of them are LUT analysis based algorithms, and the other is a LUT and interconnect analyses based algorithm.

5.4.1 IPF Algorithm Framework

Figure 5.2 shows the IPF algorithm framework. Given a mapped netlist (see Figure 5.2(a)), the IPF algorithms fill SDC bits to reduce error propagations in the netlist. The framework has two stages. First, we perform a logic simulation to collect all SDC bits in the netlist (see Figure 5.2(b) and (c)). Second, different IPF algorithms are applied for SDC bits according to different reliability requirements (see in Figure 5.2(d)).

In the first stage, a window based logic simulation is performed to collect SDC bits. The basic idea is to perform an exhaustive logic simulation on a selected window instead of the full circuit, to collect the SDC bits which are not accessible during the logic simulation. Compared to a full-circuit simulation, the collected SDC bits are underestimated. In this chapter, we adopt Cong and Minkovich's work [CM10] to collect SDC bits. The features of their work are as follows, (a) when choosing the window, the priority is given to the windows covering the most nodes given a bounded input number; (b) overlapping windows are used to minimize the controllability set, i.e., making the SDC bit collection a tight lower bound compared to a full-circuit simulation.

In the second stage, different techniques are adopted according to different objectives as follows.

- LUT analysis based: Based on the criticalities (defined in Section 5.4.2.2) of LUT configuration bits, two IPF algorithms are proposed. They are efficient and effective on the failure rate reduction due to SEUs on LUTs, but not as good at SEUs on interconnects.
- LUT and interconnect analyses based: The algorithm is based on SDC bit preferences (defined in Section 5.4.3.2) for masking errors introduced by SEUs on LUTs and interconnects. Although it has runtime overhead due to interconnect analysis compared with the two LUT analysis based algorithms, it outperforms them in terms of the failure rate reduction when considering SEUs on both LUTs and interconnects. That is because they work on masking errors in the LUT under test, but this algorithm works on tolerating errors on all the possible configuration bits in the fan-in cones.

5.4.2 LUT Analysis Based IPF Algorithms

5.4.2.1 Algorithm Overview

Given the SDC bit collection, the two LUT analysis based IPF algorithms analyze the criticalities of LUT configuration bits, and leverage SDC bits to mask errors introduced by SEUs on LUTs. Algorithm 2 shows the pseudo code for the proposed algorithms. For each LUT, every configuration bit is injected an SEU at a time (see Algorithm 2 Line 3). Different from the window based logic simulation for the SDC collection, a full circuit logic simulation is carried out for the critical-

Algorithm 2 LUT analysis based IPF Algorithms

```
Require: A mapped netlist with the SDC bit collection;
Ensure: An enhanced netlist;
 1: for Each LUT in an anti-topological order do
 2:     for Each configuration bit of the LUT do
 3:         (1) Inject one SEU;
 4:         (2) Propagate signal sequence in the fan-in and fan-out cone of the LUT;
 5:         (3) Calculate the criticalities of LUT configuration bits;
 6:     end for
 7:     if Based on critical configuration bits then
 8:         Perform the algorithm in Section 5.4.2.3;
 9:     end if
10:     if Based on critical outputs then
11:         Perform the algorithm in Section 5.4.2.4;
12:     end if
13: end for
```

ity analysis of the LUT configuration bits (see Algorithm 2 Line 4 and 5). Based on the criticalities, one of in-place techniques is employed to reconfigure the SDC bits for each LUT (see Algorithm 2 Line 6-9).

The LUT enhancement is performed in an anti-topological order to ensure that LUTs can be enhanced independently, proved by Theorem 1. The independent optimization enables the two LUT analysis based IPF algorithms to work in an incremental manner. Furthermore, the full circuit analysis guarantees the accuracy of the criticalities of LUT configuration bits. Therefore, the criticality update during optimization is performed in an incremental and accurate way. In the following section, we introduce the definition and the calculation of the criticalities.

Lemma 1. *In a normal situation, SDC bits do not affect the functionality of a design due to the inaccessibility. However, in a faulty circuit, SDC bits can be hit and outputted, changing the functionality of the design.*

Theorem 1. *In the two LUT analysis based IPF algorithms, LUTs can be enhanced independently if the enhancement is performed in an anti-topological order.*

Proof. Suppose the LUT that has an SEU is LUT A. The fan-in cone of LUT A can be considered as a normal circuit, while the fan-out cone can be considered as a faulty circuit due to introducing the SEU in LUT A. Based on Lemma 1, flipping SDC bits in the fan-in cone does not affect the functionality and the criticality calculation. However, changing SDC bits in the fan-out cone may affect them. In other words, if the LUT enhancement is performed in a topological order, the criticalities of bits in preceding LUTs may be changed. The two LUT analysis based IPF algorithms rely on the criticalities of configuration bits. Therefore, only the anti-topological order can guarantee that LUTs can be optimized independently. $\square$

5.4.2.2 Evaluation of the Impact of SEUs

In the literature, there are two kinds of methods for the evaluation of the impact for SEUs. One is seeking for the analytical solution, and the other is based on logic simulation. The analytical based method [AT05] cannot be practically used for large designs due to the lack of support for signal convergence of both faulty and non-faulty paths. In this chapter, we adopt the prevailing logic simulation based evaluation used in the previous work [MZS06, KPM07, CM10, JLF11].

A logic simulation computes the value of the outputs of internal nodes and the primary outputs of a logic network, given a set of input vectors. One run of simulation propagates one set of vector through the network. Its complexity is in linear in the network size. Targeting the impact of an SEU on a particular configuration bit, we flip the value of the configuration bit, and perform simulations, to see what the percentage of input vectors is resulting in the changes in primary outputs of the logic network. In other words, based on the aforementioned simulation, utilizing Equations 5.1 and 5.2, we derive the approximate failure rate for each configuration bit, used as the criticality of the configuration bit. According to Luckenbill's experiments [LLH10], 1K randomly generated input vectors yield a close estimation to the full input vectors with a mean error of 1%, and 128K input vectors reduces the error to 0.3%. We perform the same 100K simulations for each configuration bit to evaluate the impact of SEUs in this chapter. Considering that our algorithms achieve up to 37% failure rate reduction, the error is sufficient for the evaluation of the impact of SEUs.

To ensure the efficiency of the simulation, in our implementation, we employ the following techniques,

- Performing simulations in a bit-parallel manner, i.e., simulating 32 or 64 runs at the same time.
- Performing simulations in a incremental style, i.e., except for the first 100K

simulations, we only need to re-simulate and propagate the errors in the fan-in and fan-out cone of the current node, and stop the simulation once there is no change of outputs of nodes during the propagation.

- Using And-Invert Graph [Mis11] representation for the logic network. Simulating an AIG node involves bitwise operations on the simulation information of the fan-ins.

5.4.2.3 Critical Configuration Bit Based

For each SDC bit, only a limited number of configuration bits can be augmented by the SDC bit. They are the n LUT configuration bits with 1 hamming distance from the SDC bit, where n is the number of inputs of the LUT. In Figure 5.3(a), suppose C_{010} is an SDC bit, it can be used to augment C_{000}, C_{011}, and C_{110}.

In this algorithm, an SDC bit is filled with the same value as that of the bit with the highest criticality in the LUT under test. As a result, whenever an SEU is in the fan-in cone, resulting in the SDC bit being accessed instead of that bit, the output is still correct. In Figure 5.3 (a), C_{011} has the highest criticality 5%. As a result, SDC bit C_{010} is filled with 1.

5.4.2.4 Critical Output Based

The algorithm focuses on which output results in more errors if not corrected. In other words, an SDC bit is assigned 1 or 0 according to the sum of the criticalities of all the candidate bits in the On set and Off set in the LUT. In Figure 5.3 (b), although the criticality of C_{011} is the highest, C_{010} is assigned 0, because the sum of the criticalities of candidate bits in the Off set is 6%, higher than that of the On set. Note that the more critical output is not necessarily the more frequent output. The former is based on the criticality calculation in a faulty circuit logic simulation, and the latter is based on a logic simulation for a normal circuit. The algorithm is focused on the more critical outputs.

5.4.3 LUT and Interconnect Analyses based IPF Algorithm

5.4.3.1 Algorithm Overview

Given the SDC bit collection, the LUT and interconnect analyses based IPF algorithm assumes that SEUs can be on LUTs and interconnects. In Figure 5.4, given a mapped netlist with the SDC bit collection, the SDC bit preference analysis is performed on LUTs and interconnects separately. After combining preferences from LUTs and interconnects analysis, SDC bits are filled, and an enhanced netlist is dumped.

The key difference between the LUT and interconnect analyses based algorithm and the "critical output based" IPF algorithm in Section 5.4.2.4 is that, the latter calculates the criticality of a configuration bit regardless of whether an SDC bit is hit or not, while the former only counts when an SDC bit is actually hit and outputted to mask errors, i.e., the SDC bit is in effect. In the following sections, SDC bit preferences for masking errors are introduced.

5.4.3.2 SDC Bit Preferences for Masking Errors Introduced by SEUs on LUTs

In order to mask errors by assigning the value to SDC bits, we want to know, in a faulty circuit simulation, how many times the propagated errors could be masked if the SDC bits are pre-set as 0 (or 1), i.e., the SDC bit preferences to be pre-set as 0 (or 1). The more preferable value is assigned to the SDC bit for SEU mitigation. Here is an example illustrating the calculation of the SDC bit preference for 0 shown Figure 5.5. Supposing that the output sequence of a LUT is 0101 without the impact of SEUs, denoted as G, and the output sequence under SEUs is 0110, denoted as F. Then, the difference of the two output sequence is 0011, denoted as D. For a specific SDC bit on the LUT, supposing that the SDC bit is hit at only the input vectors corresponding to the last two values in the output sequence, i.e., 0011, denoted as H. For the third column in shade in Figure 5.5, there is a difference between G and F, and G outputs 0. At the same time, an SDC bit is hit according to H. The error can be tolerated if the SDC bit is pre-set to 0. Therefore, the SDC bit preference for 0 increases by 1. In summary, calculations for SDC bit preferences to mask errors introduced by SEUs on LUTs are as follows,

$$\begin{aligned} 1 - preference &= count\_1\{H\&D\&G\} \\ &= count\_1\{H\&(G \wedge F)\&G\} \\ &= count\_1\{H\&\bar{F}\&G\} \end{aligned} \tag{5.3}$$

$$\begin{aligned} 0 - preference &= count\_1\{H\&D\&\bar{G}\} \\ &= count\_1\{H\&(G \wedge F)\&\bar{G}\} \\ &= count\_1\{H\&F\&\bar{G}\} \end{aligned} \tag{5.4}$$

In Equation (5.3) and (5.4), *count_1* counts the number of 1s in a vector. Equation (5.3) computes the chance that the output is 1 in a normal circuit, when an SDC bit is hit in a faulty circuit, and there is a difference between the output of the LUT with and without introducing an SEU. The same logic applies for Equation (5.4). Similar calculations of the SDC bit preferences can be extended to interconnect.

5.4.3.3 SDC Bit Preferences for Masking Errors Introduced by SEUs on Interconnects

In this chapter, we consider the island-style FPGA architecture discussed in Section 2.1. In this architecture, interconnects consist of local wires, connection boxes, and switch boxes. The signal routes are directed by configuration bits in the three components. Configuration bits in interconnects can be flipped by SEUs, causing SDC bits in LUTs hit and outputted. We apply a logic simulation similar to that in Section 5.4.3.2 to configuration bits in local wires, connection boxes, and switch boxes respectively, i.e., we flip a configuration bit in one of the three components once a time, and perform SDC bit preference calculations. Combining the three categories, the total bit preferences are available for masking errors introduced by SEUs on interconnects. Merging SDC bits preferences for masking errors introduced by SEUs on LUTs and interconnects, we fill SDC bits and produce the netlist with the enhanced reliability.

5.4.4 Complexity Analysis

The runtime for IPF algorithms can be broken down into three portions: the runtime for SDC bits collection, the runtime for evaluation of the impact of SEUs on configuration bits, and the time spent on reconfiguration. The reconfiguration is performed in constant time after gathering the criticalities of configuration bits. Besides, we adopt window based logic simulation for SDC bits collection and full-chip simulation for the evaluation of SEU impact. Therefore, the runtime is dominated by the time for evaluation shown in Equation 5.8. N_W is the number of simulations in 64-bit machine words. n and m are the number of nodes and the average number of configuration bits insides one node, respectively. T_1 denotes the simulation time spent on one node. T_{ini} denotes the simulation time for calculating the golden result without the impact of SEUs. T_{inc} denotes the simulation time for incrementally evaluating the impact of SEUs on configuration bits. L denotes the number of nodes to be re-simulated in each simulation during the incremental process. In summary, the computation complexity of IPF algorithms is $O(nm)$.

$$N_W = \frac{100K}{64} \tag{5.5}$$
$$T_{ini} = N_W \cdot n \cdot T_1 \tag{5.6}$$
$$T_{inc} = N_W \cdot n \cdot m \cdot T_1 \tag{5.7}$$
$$\begin{aligned} T_{total} &= T_{ini} + T_{inc} \\ &= N_W \cdot n \cdot T_1(1 + m \cdot L) \\ &= \frac{100K}{64} \cdot n \cdot T_1(1 + m \cdot L)(L \ll n) \end{aligned} \tag{5.8}$$

5.5 Experimental Results

Table 5.2: The failure rate comparison of X-Filling algorithms for SEUs on LUTs for the 6-LUT mapping

Circuits	LUT#	Failure rate reduction for SEUs on LUTs					
		Random	One	Zero	IPF		
					LUT based		LUT and interconnect based
					Critical conf bit	Critical output	
alu4	507	0.00%	8.57%	0.00%	22.86%	20.00%	14.29%
apex2	687	-7.69%	-3.85%	-7.69%	15.38%	15.38%	11.54%
apex4	594	-3.57%	0.89%	-8.04%	16.07%	15.18%	13.39%
des	556	-4.31%	-3.45%	-6.03%	9.48%	7.76%	6.90%
ex1010	668	-6.40%	-0.80%	-9.60%	9.60%	8.80%	8.00%
ex5p	384	-44.74%	-32.89%	-53.95%	7.89%	9.21%	2.63%
misex3	490	5.56%	7.41%	-3.70%	25.93%	27.78%	22.22%
pdc	1515	-24.44%	-24.44%	-31.11%	11.11%	10.00%	5.56%
seq	705	3.17%	7.94%	0.00%	20.63%	20.63%	19.05%
spla	1436	2.50%	5.00%	1.67%	15.83%	15.83%	14.17%
Average	754	-7.99%	-3.56%	-11.85%	15.48%	15.06%	11.77%

The proposed IPF algorithms are implemented in C++, and tested on PC

Table 5.3: The failure rate comparison of X-Filling algorithms for SEUs on LUTs for the 4-LUT mapping

Circuits	LUT#	Failure rate reduction for SEUs on LUTs					
		Random	One	Zero	IPF		
					LUT based		LUT and interconnect based
					Critical conf bit	Critical output	
alu4	720	0.77%	0.00%	2.31%	10.77%	10.00%	8.46%
apex2	965	-0.99%	-0.99%	-0.99%	5.94%	3.96%	1.98%
apex4	791	-2.69%	-1.24%	-2.07%	6.40%	7.02%	8.06%
des	1249	-6.98%	-6.53%	-5.18%	4.50%	4.73%	3.60%
ex1010	1103	-4.72%	-4.33%	-4.33%	2.36%	2.56%	2.17%
ex5p	541	-39.68%	-37.78%	-41.59%	6.35%	5.08%	4.44%
misex3	736	1.54%	0.51%	0.00%	11.28%	12.31%	9.74%
pdc	2210	-18.33%	-16.17%	-19.95%	9.43%	10.51%	8.36%
seq	998	0.45%	1.36%	0.45%	8.64%	9.09%	4.55%
spla	2126	3.61%	1.20%	2.41%	10.24%	10.04%	9.44%
Average	1144	-6.70%	-6.40%	-6.89%	7.59%	7.53%	4.11%

with dual core CPU E4400 @ 2.00GHz and 2.0 GB of RAM. For the ten largest combinational MCNC benchmark circuits [Yan91], we use designs mapped by the Berkeley ABC mapper [Mis11] as the baseline. All designs enhanced by our IPF algorithms have passed the functional equivalent checking by the Berkeley ABC mapper.

- First, assuming SEUs on LUT configuration bits only, we compare our algorithms with other X-Filling algorithms, which are focused on filling SDC bits to mitigate errors introduced by SEUs, to reveal the effectiveness of our algorithms.

- Second, to test our interconnect analysis engine, we compare our LUT analysis based IPF algorithms with our LUT and interconnect analyses based algorithm for SEUs on interconnects.

- Finally, the circuit level improvement and the runtime comparisons between our algorithms and another synthesis based algorithm are provided.

5.5.1 Evaluation of X-Filling Algorithms for SEUs on LUTs

Table 5.2 and 5.3 show the evaluation for six X-Filling algorithms. They all focus on filling SDC bits to mitigate errors. Compared with the baseline from the Berkeley ABC mapper, we show the effectiveness our algorithms on failure rate reductions for SEUs on LUTs.

Besides our algorithms, the three other X-Filling algorithms are described as follows. The “Random”, “One”, and “Zero” algorithms fill SDC bits with 0 or 1 randomly, with all 1, and with all 0, respectively. For our algorithms, the column “Critical conf bit” represents the algorithm that employs SDC bits to mask errors on the most critical configuration bit in Section 5.4.2.3. The column “Critical output” represents the algorithm that utilizes SDC bits to mask more critical outputs in Section 5.4.2.4. Both of the two algorithms are based on the LUT analysis. The last algorithm is the LUT and interconnect analyses based algorithm discussed in Section 5.4.3.

For each configuration bit, 100K input vectors are injected into the circuit. Table 5.2 shows that all other X-Filling algorithms, “Random”, “One” and “Zero” generate designs with worse failure rates. Compared to the baseline from the Berkeley ABC mapper, they increase failure rates by 7.99%, 3.56%, and 11.85% for the 6-LUT mapping on average. For some circuits, they increase the failure rate by greater than 50%, e.g., Circuit “ex5p”. For the 4-LUT mapping, they increase failure rates by 6.70%, 6.40%, and 6.89% in Table 5.3.

Our IPF algorithms reduce failure rates for both the 6-LUT mapping and the 4-LUT mapping. For the 6-LUT mapping, our LUT analysis based algorithms “Critical conf bit” and “Critical output” reduce failure rates by 15.48% and 15.06%. The LUT and interconnect analyses based algorithm reduces the failure rate lower than those of the two algorithms, because it takes into account SEUs on interconnects. For the 4-LUT mapping, Table 5.3 shows the same trend except that the failure rate reductions are lower than those for the 6-LUT mapping. The reason is that, designs mapped to 4-LUTs have less SDC bits according to Table 5.1. That means we have less flexibility and potentially less room to augment the reliability of designs.

Considering SEUs on LUTs, Table 5.2 and 5.3 show the failure rate comparison of the X-Filling algorithms. We conclude that our IPF algorithms produce better results than other X-Filling algorithms. The two LUT analysis based algorithms outperform the LUT and interconnect analyses based algorithm. Whether designs are mapped to 6-LUTs or 4-LUTs also affects the failure rate reductions.

5.5.2 Evaluation of the IPF Algorithms for SEUs on Interconnects

Table 5.4: The failure rate comparison of the IPF algorithms for SEUs on interconnects for the 6-LUT mapping

Circuits	LUT#	Failure rate reduction for SEUs on interconnects								
		LUT based						LUT and interconnect based		
		Critical conf bit			Critical output					
		Local	SBOX	CBOX	Local	SBOX	CBOX	Local	SBOX	CBOX
alu4	507	8.58%	8.79%	9.11%	8.31%	4.34%	47.21%	46.67%	29.83%	29.51%
apex2	687	9.99%	5.92%	10.17%	8.82%	-3.55%	13.20%	34.22%	40.08%	38.43%
apex4	594	11.17%	9.34%	21.73%	11.88%	12.16%	18.34%	51.39%	63.50%	62.20%
des	556	6.27%	-6.35%	2.34%	3.49%	-6.82%	3.16%	20.02%	12.18%	9.09%
ex1010	668	5.67%	29.15%	20.11%	-6.76%	16.07%	5.26%	45.11%	67.78%	62.10%
ex5p	384	8.59%	-1.36%	39.33%	11.77%	0.62%	42.38%	23.89%	25.10%	30.81%
misex3	490	11.32%	24.30%	20.96%	16.65%	27.43%	16.23%	63.71%	61.09%	58.16%
pdc	1515	17.92%	23.01%	22.63%	20.54%	31.72%	27.73%	53.17%	65.67%	64.79%
seq	705	10.98%	25.67%	19.53%	6.31%	22.34%	14.06%	49.40%	60.47%	51.60%
spla	1436	13.09%	25.42%	50.43%	14.44%	23.34%	46.94%	53.49%	67.32%	76.41%
Average	754	10.36%	14.39%	21.63%	9.54%	12.76%	23.45%	44.11%	49.30%	48.31%

To test our interconnect analysis engine, we compare the two LUT analysis based IPF algorithms with the LUT and interconnect analyses based algorithm for SEUs on interconnects.

We adopt the Jing's work [JLF11] to perform the interconnect evaluation for

Table 5.5: The failure rate comparison of the IPF algorithms for SEUs on interconnects for the 4-LUT mapping

Circuits	LUT#	Failure rate reduction for SEUs on interconnects								
		LUT based						LUT and interconnect based		
		Critical conf bit			Critical output					
		Local	SBOX	CBOX	Local	SBOX	CBOX	Local	SBOX	CBOX
alu4	720	0.50%	-3.71%	3.56%	2.33%	-7.93%	-2.19%	21.89%	25.26%	27.65%
apex2	965	2.33%	10.13%	-3.53%	1.02%	18.53%	-5.86%	14.59%	27.78%	14.35%
apex4	791	-2.20%	2.34%	2.27%	-1.18%	-1.69%	-2.39%	28.37%	47.73%	59.28%
des	1249	-3.22%	8.23%	-0.03%	-1.84%	0.17%	-1.28%	4.55%	10.78%	8.37%
ex1010	1103	-3.58%	3.41%	-1.91%	-1.04%	-3.38%	-7.44%	2.01%	18.91%	-14.29%
ex5p	541	14.04%	25.15%	18.92%	14.13%	20.77%	19.64%	25.68%	36.88%	35.93%
misex3	736	-1.09%	15.38%	7.55%	-0.26%	10.71%	9.54%	25.87%	44.17%	48.37%
pdc	2210	13.66%	19.18%	18.55%	8.50%	18.21%	17.67%	14.44%	45.84%	46.45%
seq	998	5.06%	6.57%	-1.56%	1.63%	-2.68%	-33.65%	29.00%	48.93%	39.75%
spla	2126	9.61%	15.30%	16.54%	6.55%	11.97%	15.28%	15.13%	44.42%	45.01%
Average	1144	3.51%	10.20%	6.04%	2.98%	6.47%	0.93%	18.15%	35.07%	31.09%

island-style FPGAs. In this architecture, interconnects are composed of local wires, switching boxes, and connection boxes. Each time, we flip a configuration bit in one of the three components, and perform a full-circuit logic simulation. For each configuration bit, 100K input vectors are injected into circuits. Considering the average number of LUTs in circuits is 754 for the 6-LUT mapping, and the number of interconnect configuration bits is roughly more than 5× compared with those of LUTs, we perform approximately 24 billion simulation runs for the interconnect evaluation. In Table 5.4 and 5.5, we label the failure rate reductions for local wires, switching boxes, and connection boxes as "Local", "SBOX", and "CBOX", respectively.

For the 6-LUT mapping in Table 5.4, the algorithm in the last three columns considering interconnects achieves a 40% failure rate reduction on average, higher than the two LUT based algorithms, which reduce failure rates by 10%. Especially, some circuits yield worse failure rates in some interconnect categories using the two LUT analysis based algorithms. For Circuit "des", the failure rate due to SEUs on switch boxes is increased by 6.35% by the “Critical conf bit”algorithm. For Circuit “ex1010”, the failure rate due to SEUs on local wires is increased by 6.76% by the “Critical output” algorithm. However, none of the designs gets failure rates downgraded in any interconnect category using the LUT and interconnect analyses based algorithm for the 6-LUT mapping. Because designs mapped to 4-LUT have less SDCs, all the algorithms yield lower failure rate reductions compared to designs mapped to 6-LUTs. The algorithm considering the interconnect impact still achieves a 15% greater improvement compared with the ones that do not.

In summary, for SEUs on interconnects, the algorithm considering the interconnect impact achieves a greater failure rate reduction for all the designs for both the 6-LUT and the 4-LUT mapping than the two LUT analysis based algorithms.

5.5.3 Evaluation of Synthesis based SEU Mitigation Techniques on the Circuit Level

In this section, we perform the circuit level evaluation for failure rates of synthesis based SEU mitigation techniques. Circuit level means that SEUs can be on LUTs and interconnects during the evaluation. We compare our IPF algorithms with the best known synthesis based in-place algorithm, the In-Place Decomposition (IPD) algorithm [LFH10]. We only present comparisons for the 6-LUT mapping considering the IPD algorithm has only 6-LUT mapping results in public.

In Figure 5.6, the X-axis lists the ten largest combinational MCNC benchmark circuits according to the area of designs, and the Y-axis lists circuit level failure rate reductions. In the figure, when the designs are small, the IPD algorithm and our LUT analysis based algorithms yield similar failure rate reductions. As the design size increases, our algorithms outperform the IPD algorithm. Our LUT and interconnect analyses based algorithm always generate a design with a better reliability compared with the IPD algorithm. For the circuit “des”, the reason why the failure rate is increased by the LUT analysis based algorithm, but the rate can be reduced by IPD algorithm and our LUT and Interconnect analyses based algorithm is that, the criticality used in LUT analysis based algorithm does not consider whether the SDC bit is actually hit and is used for logic masking; however, our LUT and interconnect analyses based algorithm takes the SDC bit hit into account when calculating the SDC bit preference, and results in the failure rate reduction

The IPD algorithm targets SEUs on LUT configuration bits, and yields a 7% circuit level failure rate reduction, although it is known for high failure rate reduction when considering SEUs on LUTs only. On the circuit level, our LUT analysis based IPF algorithms achieve a 6% higher reduction compared with the IPD algorithm, and our LUT and interconnect analyses based IPF algorithm achieves a 37% greater improvement.

5.5.4 Runtime Comparison of SEU Mitigation Techniques on the Circuit Level

Figure 5.7 presents runtime comparisons for the IPD algorithm and our IPF algorithms for the 6-LUT mapping. Similarly, the X-axis lists the ten largest combinational MCNC benchmark circuits according to the area of designs, and the Y-axis lists runtime in seconds. On average, our “Critical conf bit” and “Critical output” algorithms achieve 150× and 142× speedup compared with the IPD algorithm. Although our LUT and interconnect analyses based algorithm requires a runtime overhead for the interconnect analysis, it still achieves 7× speedup compared with the IPD algorithm. The fast synthesis time makes our IPF algorithms scalable in practice.

The reason for the short runtime is that our approaches do not search for functionally equivalent implementations, and therefore do not need time-consuming

Binary Decision Diagrams, Boolean Satisfiability [HFH08, FHH09], Integer Linear Programming [LFH10], or Set of Pairs of Functions to be Distinguished [JHM10] as adopted in other synthesis based algorithms. Furthermore, when performing the criticality analysis and SDC bit preference analysis, we adopt a 64-bit word-wise logic simulation instead of a bit-wise logic simulation.

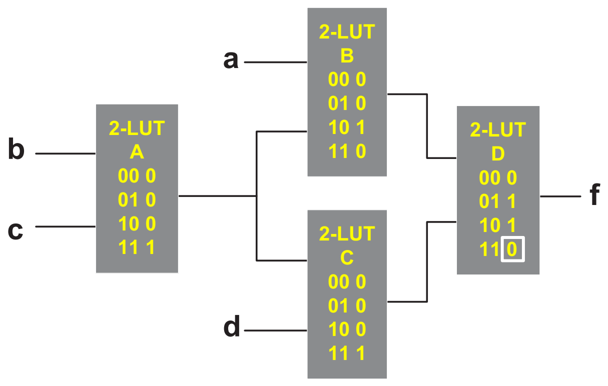
In summary, our IPF algorithms outperform the IPD algorithm in terms of both the failure rate reduction and the runtime.

5.6 Conclusions and Future Work

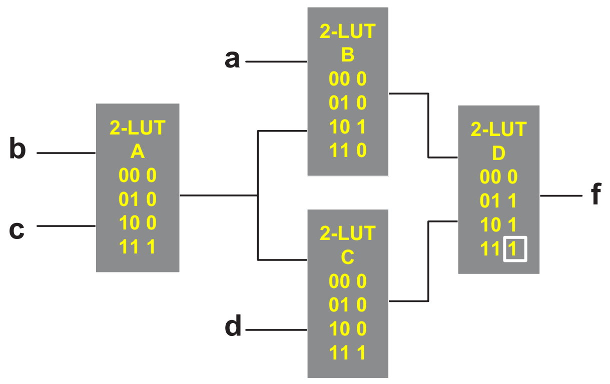
Targeting the ever-increasing SEU issue, we propose three synthesis based in-place X-Filling algorithms by exploiting don't cares to augment the reliability of designs. Compared with circuit and architecture based solutions, our algorithms are in place, and do not require area, power, performance, and design time overheads. Compared with other synthesis based algorithms, we take into account the widely accepted interconnect architecture used in VPR [BR97, LKJ11] during optimization. For the ten largest combinational MCNC benchmark circuits mapped to 6-LUTs, our approaches achieve up to a 37% greater failure rate reduction on the circuit level, and up to 150× runtime speedup, compared with the best known synthesis based in-place algorithm, the In-Place Decomposition (IPD) algorithm.

The more don't cares are reconfigured to mask errors introduced by SEUs, the greater failure rate reduction we can achieve. Increasing don't cares during synthesis can be leveraged to obtain targeted trade-off between reliability and area in the future. Furthermore, we plan to extend the IPF algorithms to handle sequential circuits. The key for this extension is to model the error propagations in sequential cycles efficiently. In addition, the impact of the technology scaling makes multiple errors introduced by SEUs a big concern. The difficulty to extend the IPF algorithms for multiple errors is to tackle the correlation between errors.

The three issues will be addressed in the future work.



(a) Failure rate=0.2031



(b) Failure rate=0.1875

$$f = ab + a\bar{c} + bc\bar{d} + ac\bar{d}$$

Figure 5.1: Given the same functionality and topology, different implementations yield different failure rates due to the assignment of the SDC bit.

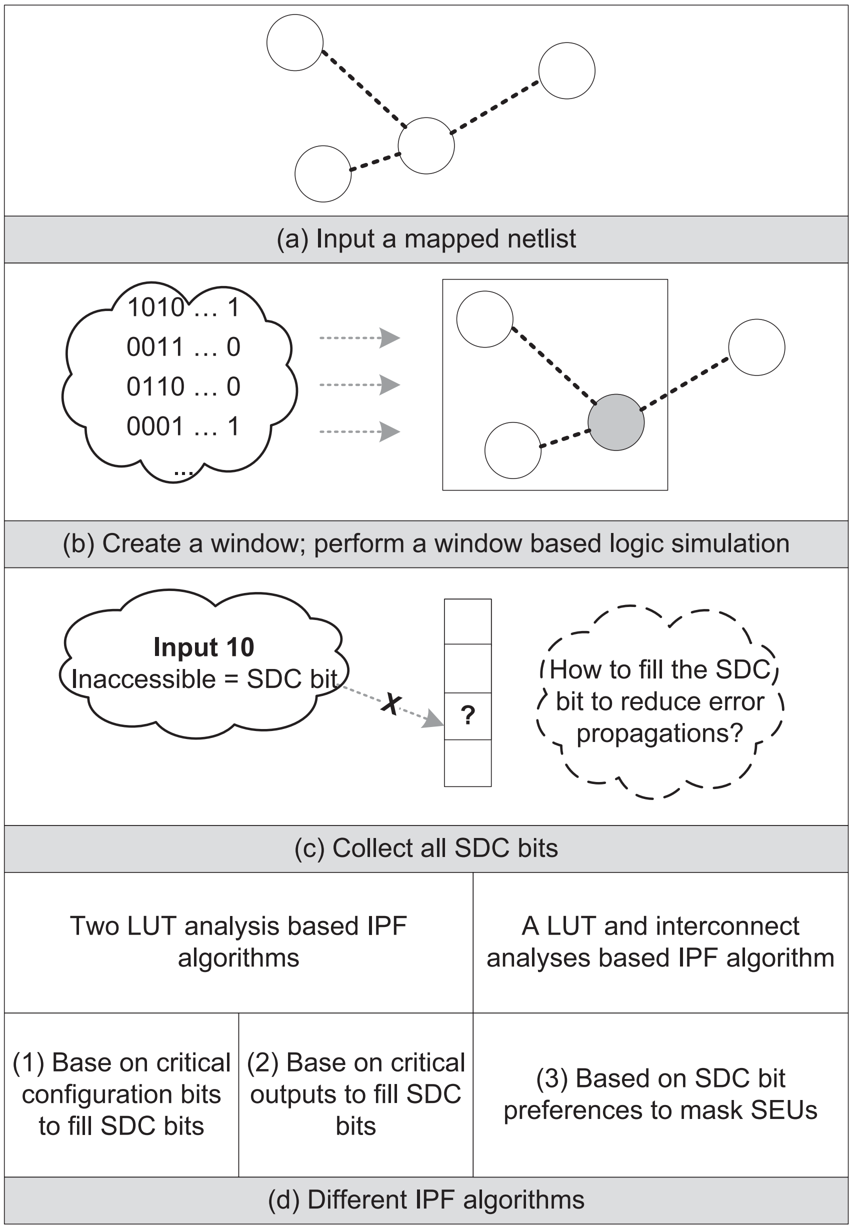

(a) Input a mapped netlist
1010 … 1
0011 … 0
0110 … 0
0001 … 1
…
(b) Create a window; perform a window based logic simulation
Input 10
Inaccessible = SDC bit
X
?
How to fill the SDC bit to reduce error propagations?
(c) Collect all SDC bits
Two LUT analysis based IPF algorithms
A LUT and interconnect analyses based IPF algorithm
(1) Base on critical configuration bits to fill SDC bits
(2) Base on critical outputs to fill SDC bits
(3) Based on SDC bit preferences to mask SEUs
(d) Different IPF algorithms

Figure 5.2: An illustration of the IPF algorithm framework

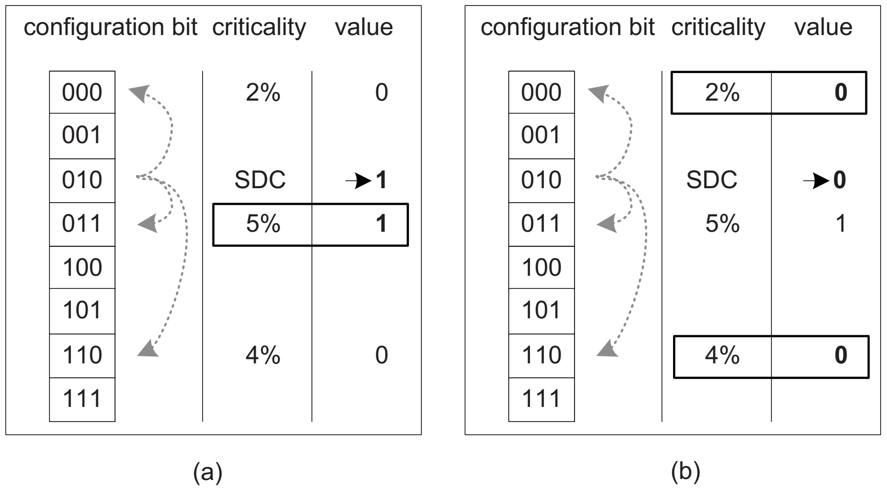


Figure 5.3: An illustration of the two LUT analysis based IPF algorithms (a) Critical configuration bit based (b) Critical output based

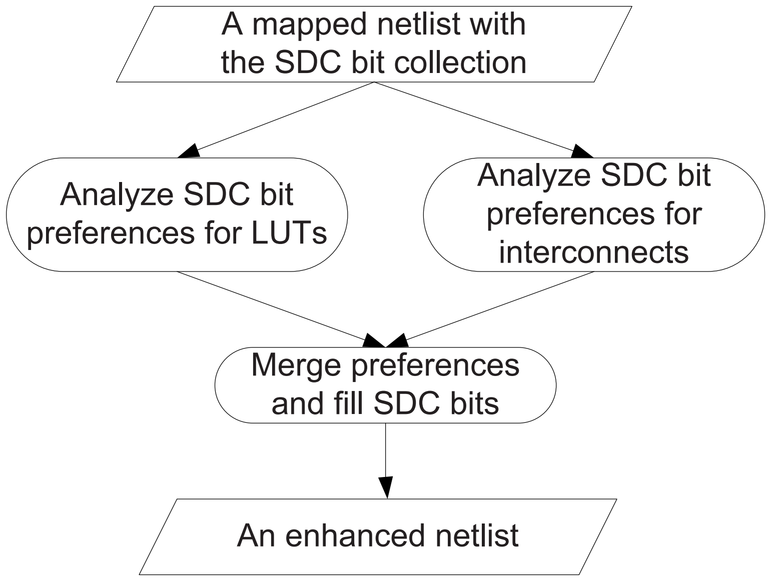

A mapped netlist with the SDC bit collection
Analyze SDC bit preferences for LUTs
Analyze SDC bit preferences for interconnects
Merge preferences and fill SDC bits
An enhanced netlist

Figure 5.4: An overview of the LUT and interconnect analyses based IPF algorithm

D	0	0	1	1
G	0	1	0	1
F	0	1	1	0
H	0	0	1	1

Figure 5.5: An example of the SDC bit preference for masking errors introduced by SEUs on LUTs

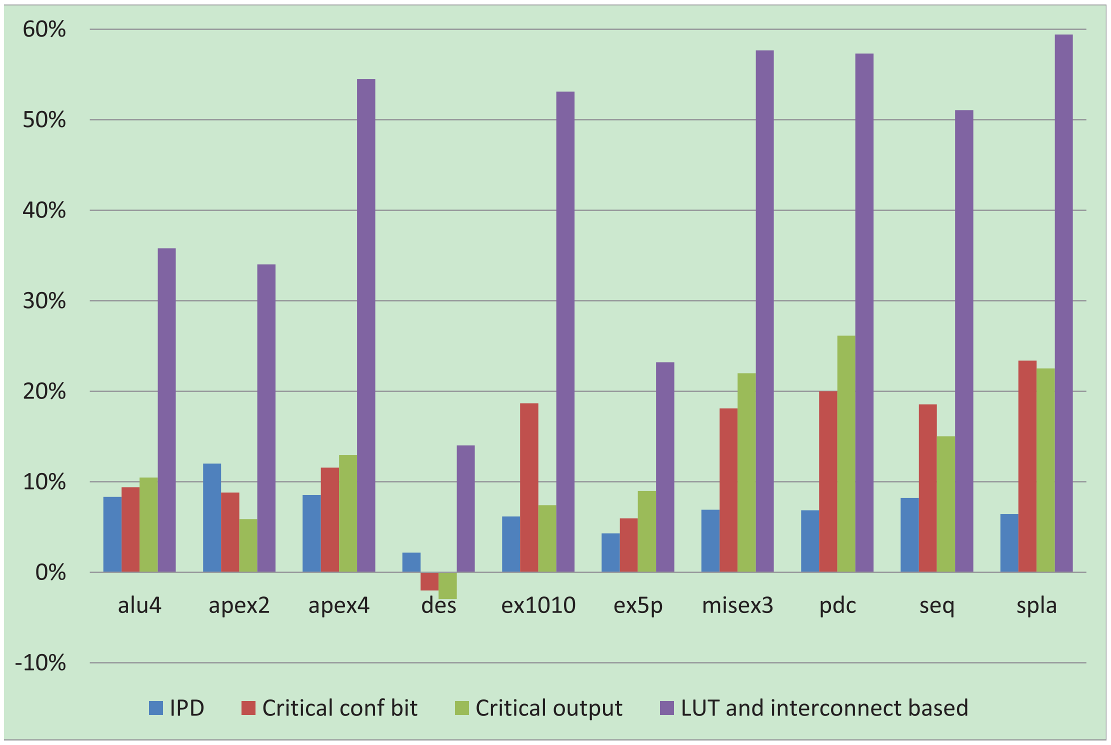


Figure 5.6: The failure rate comparison of synthesis based SEU mitigation techniques on the circuit level for the 6-LUT mapping.

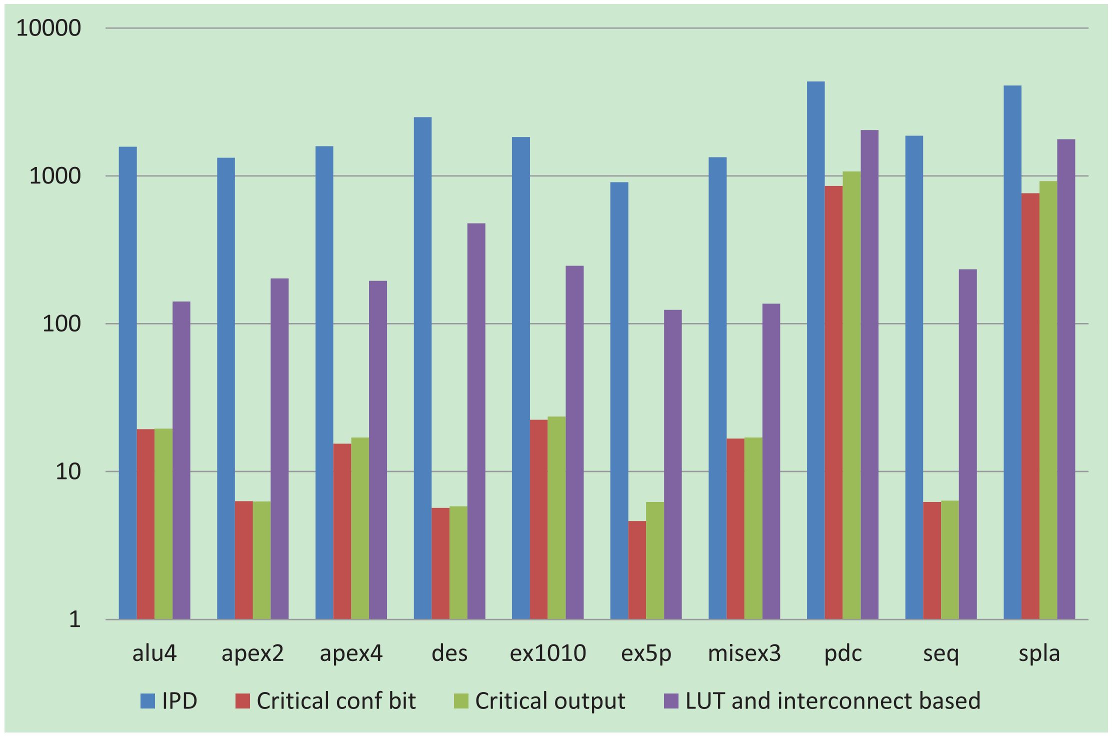


Figure 5.7: The runtime comparison of SEU mitigation techniques on the circuit level for the 6-LUT mapping.

CHAPTER 6

Conclusions and Future Work

Focusing on the reliability of FPGA based designs, to meet the industrial needs on cost (e.g., area and power overhead), CAD flow, runtime, and FPGA architecture, this dissertation proposes several novel logic synthesis algorithms. Chapter 3 presents an algorithm seeking a formal method to improve the reliability of FPGA based designs while incurring minimal area and power overhead. In Chapter 4, an in-place logic synthesis algorithm is presented so that there is no need for iterations of logic and physical synthesis during reliability optimization, i.e., no break for the current CAD flow. Chapter 5 is focused on the runtime and FPGA architecture. Three fast in-place logic synthesis algorithms targeting the modern FPGA architecture including both LUTs and interconnects, are presented.

In the effort to enhance the FPGA reliability while minimizing the area and power overhead, we formulate the reliability problem under random faults as a stochastic satisfiability (SSAT) based Boolean matching, and employs both area efficient and robust templates to rewrite the netlist (Chapter 3). During each iteration of remapping, among the solutions derived from Boolean matching, only the one that improves the reliability and incurs least overhead is chosen. It results in a 25% reduction for the failure rate with 1% fewer LUTs.

Minimizing the area and power overhead is not the only requirement from FPGA designers when they face the reliability challenge. The least modification of the netlist is desired as there is no need for iterations of logic and physical syntheses when improving the reliability. In Chapter 4 and Chapter 5, we propose in-place reconfiguration algorithms. The algorithm in Chapter 4 is focused on errors on LUTs, while the algorithms in Chapter 5 take further advantages of the error analyses on both LUTs and interconnects. Compared with the the best known in-place algorithm, the in-place decomposition (IPD) algorithm, for the ten largest combinational MCNC benchmark circuits mapped to 6-LUTs, our approaches achieve up to a 37% greater failure rate reduction, and up to 150× runtime speedup.

In the future, an interesting research direction is to address the tradeoff between logic masking and testability or verifiability, considering that introducing additional logic masking, defect-aware logic synthesis algorithms can make verification and silicon debugging tasks difficult. One possible solution is to provide proofs of correctness of transformations [KLG08]. Another interesting future research direction is to extend our algorithm to standard cell-based circuits. There is some existing work on fault-tolerant logic synthesis for standard cell designs. For example, [NS04] developed a critical-area driven technology mapping, and [KPM07] applied logic redundancy and structural restructure to mask soft errors based on a fast simulation. The CAD flow for standard cell designs can benefit from our in-place reliability enhancement algorithms.

References

[ALT] ALTERA. "Altera: QUIP for Quartus II V5.0." In *http://www.altera.com/education/univ/.*

[ALT06] ALTERA. "Altera Stratix II Features." In *http://www.altera.com/products/devices/stratix2/features/density/st2-density.html*, 2006.

[ALT12] ALTERA. "Reducing total system cost with low-power 28-nm FPGAs." In *http://www.altera.com, WP-01180-1.1*, Apr. 2012.

[AN06] Jason H. Anderson and Farid N. Najm. "Active leakage power optimization for FPGAs." *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, **25**(3), Mar. 2006.

[AT05] G. Asadi and M. B. Tahoori. "Soft Error Rate Estimation and Mitigation for SRAM Based FPGAs." In *Proc. ACM Intl. Symp. Field-Programmable Gate Arrays*, pp. 149–160, 2005.

[ATM07] Hossein Asadi, Mehdi B. Tahoori, Brian Mullins, David Kaeli, , and Kevin Granlund. "Soft error susceptibility analysis of SRAM-based FPGAs in high-performance information systems." *IEEE Trans. on Nuclear Science*, **54**(6):2714–2726, Dec. 2007.

[Bau05] Robert C. Baumann. "Radiation-Induced Soft Errors in Advanced Semiconductor Technologies." *IEEE Trans. on Device and Materials reliability*, **5**(3):305–316, Sep. 2005.

[BBB04] M. Bellato, P. Bernardi, D. Bortolato, M. Ceschia A. Candelori, A. Paccagnella, M. Rebaudengo, M. Sonza Reorda, M. Violante, and P. Zambolin. "Evaluating the effects of SEUs affecting the configuration memory of an SRAM-based FPGA." In *Design Automation and Test in Europe*, pp. 584–589, Feb. 2004.

[Ber06] Berkeley Design Technology, Inc. "BDTI Focus Report: FPGAs for DSP, Second Edition." In *BDTI Benchmarking*, 2006.

[Bid10] Nematollah Bidokhti. "SEU concept to reality (allocation, prediction, mitigation)." In *Proc. Reliability and Maintainability Symp.*, pp. 1–5, Jan. 2010.

[BKC07] Ravi Bonam, Yong-Bin Kim, and Minsu Choi. "Defect-Tolerant Gate Macro Mapping and Placement in Clock-Free Nanowire Crossbar Architecture." In *Defect and Fault-Tolerance in VLSI Systems*, pp. 161–169, Sep. 2007.

[BM97] Luca Benini and Giovanni De Micheli. "A survey of Boolean matching techniques for library binding." *ACM Transactions on Design Automation of Electronic Systems*, **2**(3):193–226, 1997.

[BM07] R. Brayton and A. Mishchenko. "Sequential Rewriting." In *Int. Workshop on Logic Synthesis*, 2007.

[BR97] V. Betz and J. Rose. "VPR: A new packing, placement and routing tool for FPGA research." In *Proc. Int. Workshop on Field-Programmable Logic and Applications*, pp. 213–222, Sep. 1997.

[BS89] R. K. Brayton and F. Somenzi. "Boolean Relations and the Incomplete Specification of Logic Networks." In *Int. Conf. on Very Large Scale Integration*, 1989.

[CH01] Jason Cong and Yean-Yow Hwang. "Boolean Matching for LUT-Based Logic Blocks With Applications to Architecture Evaluation and Technology Mapping." *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Sep. 2001.

[CKM08] Michael L. Case, Victor N. Kravets, Alan Mishchenko, and Robert K. Brayton. "Merging Nodes under Sequential Observability." In *Proc. Design Automation Conf*, pp. 540–545, 2008.

[CLH08] L. Cheng, Y. Lin, L. He, and Y. Cao. "Trace-Based Framework for Concurrent Development of Process and FPGA Architecture Considering Process Variation and Reliability." In *Proc. ACM Intl. Symp. Field-Programmable Gate Arrays*, Feb. 2008.

[CM07] Jason Cong and Kirill Minkovich. "Improved SAT-Based Boolean Matching Using Implicants for LUT-Based FPGAs." In *Proc. ACM Intl. Symp. Field-Programmable Gate Arrays*, 2007.

[CM10] J. Cong and K. Minkovich. "LUT-based FPGA technology mapping for reliability." In *Proc. Design Automation Conf*, pp. 517–522, Jun. 2010.

[DH06] Asbjoern Djupdal and Pauline C. Haddow. "Yield Enhancing Defect Tolerance Techniques for FPGAs." In *Proc. of the Int. Conf. on Military and Aerospace Programmable Logic Devices*, 2006.

[DI00] A. Doumar and H. Ito. "Design of switching blocks tolerating defects/faults in FPGA interconnection resources." In *Defect and Fault-Tolerance in VLSI Systems*, 2000.

[DP94] Serge Durand and Christian Piguet. "FPGA with Self-repair Capabilities." In *Proc. ACM Intl. Symp. Field-Programmable Gate Arrays*, 1994.

[ES] Niklas Een and Niklas Sorensso. "MiniSat." *http://minisat.se/.*

[FHH09] Z. Feng, Y. Hu, L. He, and R. Majumdar. "IPR: In-place reconfiguration for FPGA fault tolerance." In *Proc. Int. Conf. on Computer Aided Design*, pp. 105–108, Nov. 2009.

[FML04] Joseph Fabula, Jason Moore, Austin Lesea, and Saar Drimer. "The NSEU Sensitivity of Static Latch Based FPGAs and Flash Storage CPLDs." In *Proc. of the Int. Conf. on Military and Aerospace Programmable Logic Devices*, 2004.

[GA07] S. Gupta and J. Anderson. "Optimizing FPGA power with ISE design tools." *http://www.xilinx.com*, 2007.

[GCZ03] P. Graham, M. Caffrey, J. Zimmerman, P. Sundararajan, E. Johnson, and C. Patterson. "Consequences and categories of SRAM FPGA configuration SEUs." In *Proc. of the Int. Conf. on Military and Aerospace Programmable Logic Devices*, 2003.

[HAW05] O. Heron, T. Arnaout, and H.J. Wunderlich. "On the reliability evaluation of SRAM-based FPGA designs." In *Proc. Int. Conf. Field-Programmable Logic and Applications*, 2005.

[HFH08] Y. Hu, Z. Feng, L. He, and R. Majumdar. "Robust FPGA resynthesis based on fault-tolerant Boolean matching." In *Proc. Int. Conf. on Computer Aided Design*, pp. 706–713, Nov. 2008.

[HSM07] Yu Hu, Victor Shih, Rupak Majumdar, and Lei He. "Exploiting Symmetry in SAT-Based Boolean Matching for Heterogeneous FPGA Technology Mapping." In *Proc. Int. Conf. on Computer Aided Design*, 2007.

[HSM08] Yu Hu, Victor Shih, Rupak Majumdar, and Lei He. "FPGA Area Reduction by Multi-Output Function Based Sequential Resynthesis." In *Proc. Design Automation Conf*, 2008.

[HTA94] Neil J. Howard, Andrew M. Tyrrell, and Nigel M. Allinson. "The Yield Enhancement of Field-programmable Gate Arrays." In *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 1994.

[JA07] Mandar Joshi and Waleed Al-Assadi. *Development and Analysis of Defect Tolerant Bipartite Mapping Techniques for Programmable cross-points in Nanofabric Architecture.* Springer Netherlands, 2007.

[JCG03] E. Johnson, M. Caffrey, P. Graham, N. Rollins, and M. Wirthlin. "Accelerator validation of an FPGA SEU simulator." *IEEE Trans. on Nuclear Science*, **50**(6):2147–2157, Dec. 2003.

[JHM10] M. Jose, Y. Hu, R. Majumdar, and L. He. “Rewiring for robustness.” In *Proc. IEEE/ACM Design Automation Conf.*, pp. 469–474, Jun. 2010.

[JLF11] N.F. Jing, J.-Y. Lee, Z. Feng, W.F. He, Z.G. Mao, S.-J Wen, R. Wong, and L. He. “Quantitative SEU fault evaluation for SRAM-based FPGA architectures and synthesis algorithms.” In *Proc. Int. Conf. Field-Programmable Logic and Applications*, pp. 282–285, Sep. 2011.

[JLF12] Naifeng Jing, Ju-Yueh Lee, Zhe Feng, Weifeng He, Zhigang Mao, and Lei He. “SEU Fault Evaluation and Characteristics for SRAM-based FPGA Architectures and Synthesis Algorithms.” *ACM Trans. on Design Automation of Electronics Systems*, **18**(13), Dec. 2012.

[KLG08] S. Kundu, S. Lerner, and R. Gupta. “Validating High Level Synthesis.” In *Computer Aided Verification*, pp. 459–472. Springer, 2008.

[KPM07] Smita Krishnaswamy, Stephen M. Plaza, Igor L. Markov, and John P. Hayes. “Enhancing Design Robustness with Reliability-aware Resynthesis and Logic Simulation.” In *Proc. Int. Conf. on Computer Aided Design*, pp. 149–154, 2007.

[KPM09] S. Krishnaswamy, S. Plaza, I. Markov, and J. Hayes. “Signature-based SER Analysis and Design of Logic Circuits.” *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 2009.

[KTR08] Ian Kuon, Russell Tessier, and Jonathan Rose. “FPGA Architecture: Survey and Challenges.” *Foundations and Trends in Electronic Design Automation*, **2**:135–253, Feb. 2008.

[LFH10] J.-Y. Lee, Z. Feng, and L. He. “In-place decomposition for robustness in FPGA.” In *Proc. IEEE/ACM Int. Conf. on Computer-Aided Design*, pp. 143–148, Nov. 2010.

[LH07] Yan Lin and Lei He. “Device and Architecture Cocurrent Optimization for FPGA Transient Soft Error Rate.” In *Proc. Int. Conf. on Computer Aided Design*, November 2007.

[LHM09] J.-Y. Lee, Y. Hu, and R. Majumdar. “Simultaneous test pattern compaction, ordering and x-filling for testing power reduction.” In *Proc. Int. Symp. Quality Electronic Design*, pp. 702–707, Mar. 2009.

[Lit99] M. Littman. “Initial experiments in stochastic satisfiability.” In *Proc. of the national conf. on Artificial intelligence*, pp. 667–672, 1999.

[LKJ11] J. Luu, I. Kuon, P. Jamieson, T. Campbell, A. Ye, W.M. Fang, K. Kent, and J. Rose. “VPR 5.0: FPGA CAD and architecture exploration tools with single-driver routing, heterogeneity and process scaling." *ACM Trans. Reconfigurable Technology and Systems*, **4**(32), Dec. 2011.

[LLH10] Samuel Luckenbill, Ju-Yueh Lee, Yu Hu, Rupak Majumdar, and Lei He. "RALF: Reliability Analysis for Logic Faults - An Exact Algorithm and Its Applications." In *Design, Automation and Test in Europe Conference and Exhibition*, pp. 783–788, 2010.

[LMP01] M.L. Littman, S.M. Majercik, and T. Pitassi. "Stochastic Boolean Satisfiability." *J. Autom. Reasoning*, **27(3)**:251–296, 2001.

[LSB05a] A. Ling, D. Singh, and S. Brown. "FPGA Logic Synthesis using Quantified Boolean Satisfiability." In *SAT 2005 Springer LNCS Vol 3569*, pp. 444–450, 2005.

[LSB05b] A. Ling, D. Singh, and S. Brown. "FPGA technology mapping: a study of optimality." In *Proc. Design Automation Conf*, 2005.

[LV62] R. E. Lyions and W. Vanderkulk. "The use of triple modular redundancy to improve computer reliability." *IBM J. Res. Dev.*, **6**(2):200–209, Apr. 1962.

[LWG97] E. Lehman, Y. Watanabe, J. Grodstein, and H. Harkness. "Logic decomposition during technology mapping." *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, **16**(8):813–833, Aug 1997.

[MB05a] Stephen M. Majercik and Byron Boots. "DC-SSAT: A Divide-and-Conquer Approach to Solving Stochastic Satisfiability Problems Efficiently." In *Proc. of the national conf. on Artificial intelligence*, pp. 416–422, 2005.

[MB05b] Alan Mishchenko and Robert K. Brayton. "SAT-Based Complete Don't-Care Computation for Network Optimization." In *Design Automation and Test in Europe*, pp. 412–417, 2005.

[MBC08] A. Mishchenko, R. K. Brayton, and S. Chatterjee. "Boolean factoring and decomposition of logic networks." In *Proc. Int. Conf. on Computer Aided Design*, 2008.

[MCB05] Alan Mishchenko, Satrajit Chatterjee, and Robert Brayton. "DAG-Aware AIG Rewriting." In *Proc. Design Automation Conf*, 2005.

[Meh12] N. Mehta. "Xilinx redefines power, performance, and design productivity with three innovative 28 nm FPGA families: Virtex-7, Kintex-7, and Artix-7 devices." In *http://www.xilinx.com, WP373 (v1.3.1)*, May 2012.

[MER05] Shubu Mukherjee, Joel Emer, and Steven. K Reinhardt. "Radiation-Induced Soft Errors: An Architectural Perspective." In *11th International Symposium on High-Performance Computer Architecture*, 2005.

[Mic91] G. De Micheli. "Synchronous logic synthesis: algorithms for cycle-time minimization." *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 1991.

[Mis11] A. Mishchenko. "ABC: A System for Sequential Synthesis and Verification." In *http://www.eecs.berkeley.edu/alanmi/abc/*, Feb. 2011.

[MSB91] S. Malik, E. Sentovich, R. Brayton, and A. Sangiovanni-Vincentelli. "Retiming and resynthesis: Optimizing sequential networks with combinational techniques." *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, **10**:74–84, 1991.

[MSZ05] S. Mitra, N. Seifert, M. Zhang, Q. Shi, and K.S. Kim. "Robust system design with built-in soft-error resilience." *IEEE Computer*, **38**(2):43–52, Feb. 2005.

[MWK03] A. Mishchenko, X. Wang, and T. Kam. "A new enhanced constructive decomposition and mapping algorithm." In *Proc. Design Automation Conf*, 2003.

[MZS06] A. Mishchenko, J. Zhang, S. Sinha, J. Burch, R. Brayton, and M. Chrzanowska-Jeske. "Using simulation and satisfiability to compute flexibilities in Boolean networks." *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, **25**(5):743–755, May 2006.

[Nae05] Helia Naeimi. "A Greedy Algorithm for Tolerating Defective Crosspoints in NanoPLA Design." In *Master Thesis, California Institute of Technology*, 2005.

[Neu56] J. von Neumann. "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components." In C.E. Shannon and J. McCarthy, editors, *Automata Studies*, pp. 43–98. Princeton Univ. Press, 1956.

[NS04] A. Nardi and A. L. Sangiovanni-Vincentelli. "Logic Synthesis for Manufacturability." *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 2004.

[Pap85] C. Papadimitriou. "Games against nature." *Journal of Computer and Systems Sciences*, **31**:288–301, 1985.

[PCM07] S. Plaza, K-H. Chang, I. Markov, and V. Bertacco. "Node Mergers in the Presence of Don't Cares." In *Proc. Asia South Pacific Design Automation Conf.*, pp. 414–419, 2007.

[Roo04] R. Roosta. "A comparison of radiation-hard and radiation-tolerant FPGAs for space applications." *NASA Electronic Parts and Packaging Program, JPL D-31228*, Dec. 2004.

[RRV02] M. Rebaudengo, M.S. Reorda, and M. Violante. "Simulation-Based Analysis of SEU Effects on SRAM-based FPGAs." In *Proc. Int. Conf. Field-Programmable Logic and Applications*, pp. 607–615, 2002.

[SMC01] F. Sturesson, S. Mattsson, C. Carmichael, and R. Harboe-Sorensen. "Heavy ion characterization of SEU mitigation methods for the Virtex FPGA." In *6th European Conf. on Radiation and Its Effects on Components and Systems*, pp. 285–291, Sep. 2001.

[SRK04] P.K. Samudrala, J. Ramos, and S. Katkoori. "Selective triple modular redundancy (STMR) based single-event upset (SEU) tolerant synthesis for FPGAs." *IEEE Trans. on Nuclear Science*, **51**(5):2957–2969, Oct. 2004.

[SVB06] Sean Safarpour, Andreas Veneris, Gregg Baeckler, and Richard Yuan. "Efficient SAT-based Boolean Matching for FPGA Technology Mapping." In *Proc. Design Automation Conf*, 2006.

[Tam06] S. Tam. "Single error correction and double error detection." In *http://www.xilinx.com, XAPP645 (v2.2)*, Aug. 2006.

[XIL12a] XILINX. "7 Series FPGAs Overview." In *http://www.xilinx.com*, May 2012.

[XIL12b] XILINX. "Radiation-hardened, space-grade Virtex-5QV device overview." In *http://www.xilinx.com, DS192 (v1.3)*, Mar. 2012.

[XIL13] XILINX. "Xilinx Stays a Generation Ahead with Multiple 20nm Firsts." In *http://press.xilinx.com/2013-01-30-Xilinx-Stays-a-Generation-Ahead-with-Multiple-20nm-Firsts*, Jan. 2013.

[XPC05] Zhong. Xiu, David. A. Papa, Philip. Chong, A. Kuehlmann, Rob A. Rutenbar, and Igor L. Markov. "Early Research Experience With OpenAccess Gear:." In *Proc. Int. Symp. on Physical Design*, 2005.

[Yan91] S. Yang. "Logic Synthesis and Optimization Benchmarks, Version 3.0." Technical report, Microelectronics Center of North Carolina (MCNC), 1991.

[YL05] A. J. Yu and G. G. Lemieux. "Defect-tolerant FPGA switch block and connection block with fine-grain redundancy for yield enhancement." In *Proc. Int. Conf. on Field-Programmable Logic and Applicationes*, 2005.

[YSN96] Shigeru Yamashita, Hiroshi Sawada, and Akira Nagoya. "A new method to express functional permissibilities for LUT based FPGAs and its applications." In *Proc. Int. Conf. on Computer Aided Design*, 1996.