University of Alberta

Exploring Application-level Fault Tolerance for Robust Design Using FPGA

by

Jing Chen

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer, Microelectronic Devices, Circuits and Systems

Department of Electrical and Computer Engineering

© Jing Chen
Fall 2012
Edmonton, Alberta

Permission is hereby granted to the University of Alberta Libraries to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only. Where the thesis is converted to, or otherwise made available in digital form, the University of Alberta will advise potential users of the thesis of these terms.

The author reserves all other publication and other rights in association with the copyright in the thesis and, except as herein before provided, neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatsoever without the author's prior written permission.

To my dear parents, Huihua Jia and Anju Chen,
and my dear husband, Lintao Cui,
I would never achieve this without their endless love and support.

Abstract

Single Event Upset has become an increasingly important issue for SRAM-based Field Programmable Gate Arrays. To mitigate these soft errors, most of existing works focused on utilizing logic-level flexibilities to improve circuit reliability. However, we notice that from an application's perspective, there exist higher-level flexibilities. This kind of application-level fault tolerance can be useful from two aspects: one is by directly modifying algorithms (algorithm-based fault tolerance), and the other is by mapping algorithm properties into logic level (algorithm-mapping fault tolerance).

In this thesis, we perform two case studies to analyze the impact of both categories of application-level fault tolerance on circuit reliability, and explore their linkages to the logic-level fault tolerance. With an enhanced algorithm considering algorithm-based fault tolerance, the error rate for the matrix multiplication can be reduced by 18x. Moreover, by mapping algorithm properties into logic level, we achieve 3x improvement in circuit reliability for the discrete convolution.

Acknowledgements

I am deeply obliged to my supervisor Prof. Dr. Yu Hu from the Electrical and Computer Engineering Department of University of Alberta whose help, stimulating suggestions and encouragement helped me in all the time of research for this thesis.

My former colleague, now serving as a postdoctoral researcher in School of Electrical and Electronic Engineering at Nanyang Technological University, Dr. Chun Zhang supported me in my research work. My colleague Pengfei Zhu at the Electrical and Computer Engineering Department of University of Alberta also gave me some hint when I conducted my research. I want to thank them for all their help, support, interest and valuable hints. Especially I am indebted to Prof. Dr. Jie Han from the Electrical and Computer Engineering Department of University of Alberta as my course instructor as well as my committee member, whose great course material gave me great help and inspiration to my thesis. I also want to thank Prof. Dr. Guohui Lin from the Computer Science Department of University of Alberta as my committee member to help evaluate this thesis.

Especially, I would like to thank my parents, who have always been supportive for my education and my growing up with their endless love. I would like to give my special thanks to my dear husband Lintao Cui, who also is my colleague, whose patient love enabled me to complete this work and who also looked closely at the final version of the thesis for English style and grammar, correcting both and offering suggestions for improvement.

Table of Contents

1 Introduction **1**

1.1 Motivation 1

1.2 Contributions 3

1.3 Outline of thesis 4

2 Preliminaries **5**

2.1 Radiation Effects on Integrated Circuits 5

2.1.1 SEU in Digital Logic Circuit 6

2.1.2 SEU Classification 8

2.2 Introduction of FPGA 9

2.2.1 FPGA Architecture Overview 10

2.2.2 FPGA Design Flow 13

2.3 SEU Effects on SRAM-based FPGAs 16

2.4 SEU Mitigation Techniques for SRAM-based FPGAs 20

2.4.1 Architectural Mitigation 22

2.4.2 High Level Mitigation 24

2.5 Logic-Level Fault Model 26

2.6 LUT-based Boolean Network 27

2.6.1 Boolean functions and representations 27

2.6.2 ABC: A System for Sequential Synthesis and Verification . 29

3 Algorithm-based Fault Tolerance **32**

3.1 Characteristics of ABFT 32

3.2 Impact on Circuit Reliability 33

3.3 Case Study of ABFT: Matrix Multiplier 34

4 Algorithm Mapping Fault Tolerance **39**

4.1 Measurement Introduced Fault Model (M-FM) 39

4.2 Impact on Circuit Reliability 40

4.3 Case study of AMFT: Discrete Convolution 42

5 Conclusions **45**

5.1 Future Research Directions 45

Bibliography **46**

List of Tables

2.1 A 3-input Truth Table 13

3.1 Comparison between Fault-tolerant and Normal Matrix Multiplier . 37

4.1 Comparison between L-FM and M-FM for 32-Bit Adder 41

4.2 Comparison between L-FM and M-FM Based Partial TMR 44

List of Figures

1.1 A System Block Diagram to Illustrate Two Categories of AFT and the Relationship between AFT and Logic-level Fault Tolerance . . . 3

2.1 SEU in Digital Logic Circuit [3] 6
2.2 SEU in a SRAM Memory Cell [3] 7
2.3 SET in a Combinational Logic Circuit [3] 8
2.4 Xilinx Virtex FPGA Architecture Overview [22] 10
2.5 One Slice Structure [5] 11
2.6 Routing inside FPGA [22] 12
2.7 A 3-input Look-up Table 13
2.8 FPGA Design Flow 14
2.9 Mux Select Failure [5] 17
2.10 PIP Failure [5] 17
2.11 Buffer Failure [5] 18
2.12 LUT Value Change [5] 19
2.13 Control Bits Change [5] 20
2.14 Fault-tolerant Design for SRAM-based FPGA [3] 21
2.15 Resistor Hardened Memory Cell [3] 23
2.16 A Memory Row protected by Hamming and RS code [3] 23
2.17 Hamming and RS Code Protected Memory Architecture [3] 24
2.18 Triple Modular Redundancy [3] 25
2.19 Xilinx Virtex Scrubbing Scheme [3] 26
2.20 Illustration definitions of PIs/POs, fanins/fanouts and Transitive fanin/fanout cone 28

2.21 ABC Territory 29
2.22 4-bit Ripple Adder Network structure visualized by ABC 30

3.1 Theory of Fault Tolerant Matrix Multiplier 36
3.2 Error Detection and Correction 37
3.3 Criticality Reduction using Partial TMR for Normal Matrix Multiplier 37

4.1 Criticality of Different SRAM Bits in 32-Bit Adder. The Bits are Ordered Based on Fanin Cones from LSB to MSB outputs. 41
4.2 Experimental Flow 43
4.3 Difference in selected LUTs for TMR 44

List of Symbols

$G(V, E)$	Directed Graph G with vertices set V and edges set E
$C(b)(x)$	Golden output with input x and concerned bit b
$C(\bar{b})(x)$	Circuit's output with input x when researched bit b is flipped
$FI(v)$	Fanin of a node v
$FO(v)$	Fanout of a node v
$FI_T(v)$	Transitive fianin of a node v
$FO_T(v)$	Transitive fianout of a node v

Acronyms

ABC A System for Sequential Synthesis and Verification

ABFT Algorithm-Based Fault Tolerance

AFT Application-level Fault Tolerance

AIG And-Inverter Graph

AMFT Algorithm-Mapping Fault Tolerance

ASIC Application Specific Integrated Circuit

BLIF Berkeley Logic Interchange Format

BRAM Block Random Access Memory

CLB Configurable Logic Block

COTS Commercial Off-The-Shelf

DAC Digital to Analog Converter

DSP Digital Signal Processing

DLL Delay-Locked Loop

EEPROM Electrically Erasable Programmable Read-Only Memory

FPGA Field Programmable Gate Array

HDL Hardware Description Language

IC Integrated Circuit

IOB Input/Output Blocks

L-FM Logic-level Fault Model

LUT Look Up Table

MB Mega Bits

MBU Multiple Bit Upsets

M-FM Measurement-introduced Fault Model

NRE non-recurring engineering

QUIP Quartus II University Interface Program

PIP Programmable Interconnect Point

PLB Programmable Logic Block

PLD Programmable Logic Device

RAM Random Access Memory

SEE Single Event Effect

SET Single Transient Effect

SEU Single Event Upset

SOP Sum of Products

SPFD Sets of Pairs of Functions to be Distinguished

SRAM Static Random Access Memory

TMR Triple Modular Redundancy

VDSM very submicron

VHDL VHSIC Hardware Description Language

VHSIC Very High Speed Integrated Circuit

BLIF Berkeley Logic Interchange Format

PI Primary Input

PO Primary Output

Chapter 1

Introduction

1.1 Motivation

With ever advancing technology for IC fabrication, single event upset (SEU) caused by supply voltage fluctuations, electromagnetic coupling and environmental radiation has become a severe problem for circuit reliability. Compared to ASICs, SRAM-based FPGAs are more vulnerable to SEUs due to the large number of memory cells used for configuration [5] [6]. When an SEU happens in FPGAs, a specific SRAM bit is flipped and may cause permanent functional failure of the circuit before the reconfiguration or memory scrubbing is carried out.

To enhance circuit reliability, extensive studies have been carried out to mitigate the impact of SEUs during synthesis. In [38], re-convergent path-based circuit structure, which is a primary reason to introduce don't cares, is used as a template to resynthesize the circuit after technology mapping. Don't care bits are modified in [36] such that an erroneous access to theses bits won't propagate errors to the next stage. Recently, [8] suggests a window-based don't care computation method, which is further in technology mapping to select more reliable covers. The author of [42] proposes to duplicate or decompose functions and then mask errors in them by the following unused hardware carry-chain in Programmable Logic Block (PLB).

Most of the existing circuit-level fault analysis and fault-tolerant optimization were focused on the flexibilities in a logic-level network (e.g., logic don't-cares,

Boolean relations [33], Sets of Pairs of Functions to be Distinguished (SPFDs) [50], and sequential flexibilities [34]), which could mask errors from propagating to circuit's primary outputs. Moreover, Triple Modular Redundancy (TMR) [40] also can achieve fault tolerance by introducing area overhead to some extent.

However, from an application's perspective, there exist higher-level application-specific flexibilities named as application-level fault tolerance (AFT). Like the fault-tolerant matrix multiplier proposed in [39], certain algorithms have intrinsic abilities to maintain system stability in the case of internal errors. This kind of AFT, achieved by directly optimizing or redesigning the algorithm, can be viewed as algorithm-based fault tolerance (ABFT). Besides, a property of specific application can be utilized to map into logic level to improve circuit reliability, which we regard as algorithm-mapping fault tolerance (AMFT). This property can be that certain errors observed at primary outputs can sometimes be neglected as acceptable noises. For example, errors occurred in lower output bits of arithmetic circuit like Adder or Multiplier only lead to little loss in precision, which in many cases will not affect the overall system correctness. For another example, suppose we have an actuator which includes a DSP module and a DAC module that control a robotic arm. The sensitivity of robotic arm decides the precision of the interpretation for the value outputted by the DAC. Apparently, more fault tolerance is automatically achieved for the DSP module if we map the information that the DAC module has a lower precision requirement into logic level. Figure 1.1 illustrates the category of AFT and the relationship between AFT and logic-level fault tolerance.

These application-level fault tolerance flexibilities provide us with new freedoms to improve robustness of target circuit. In other words, we can enhance system reliability through algorithm redesign and/or critical circuit redefine. These two forms of AFT, ABFT and AMFT, should be considered to further achieve fault tolerance and could be viewed as more intrinsic respects to evaluate logic-level sensitivity to faults. Before we implement a design in the circuit level, the algorithmic description for this design and the specific properties of the algorithm have to some extent largely determined the logic-level sensitivity to faults.

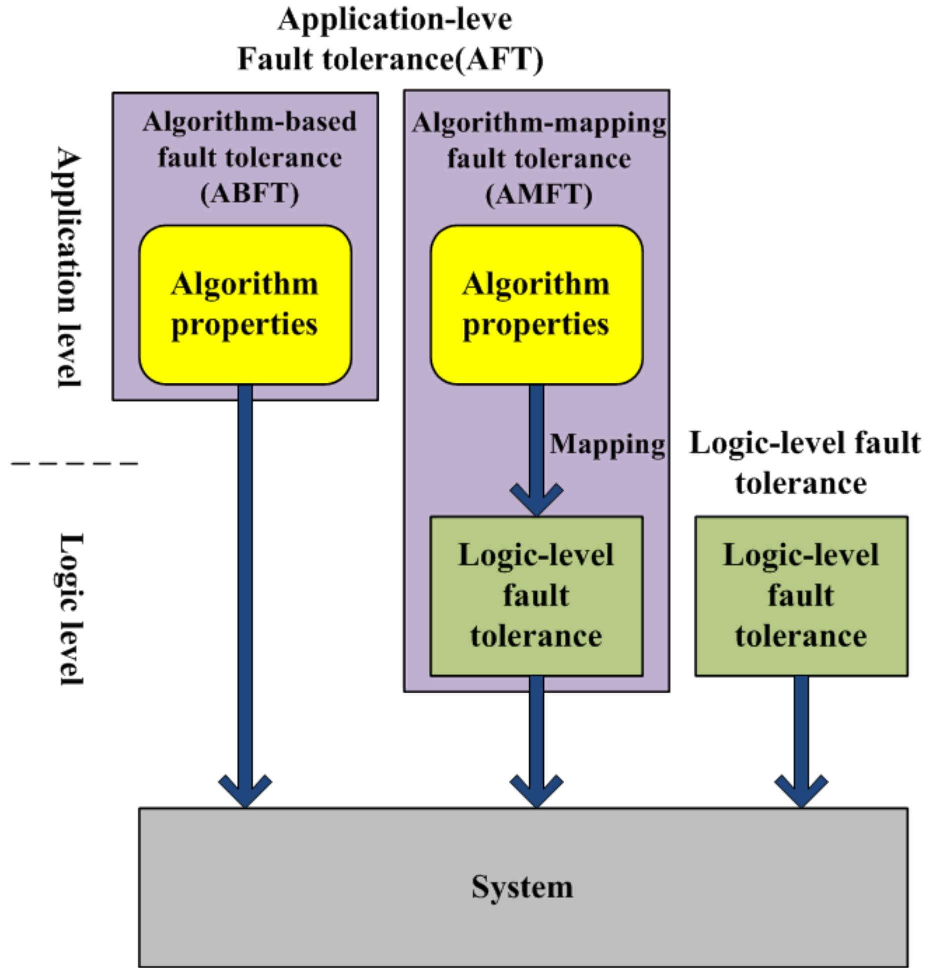


Figure 1.1: A System Block Diagram to Illustrate Two Categories of AFT and the Relationship between AFT and Logic-level Fault Tolerance

1.2 Contributions

In this thesis, we explore both forms of application-level fault tolerance and study their linkages to logic-level fault tolerance. For ABFT, though it is clear that a design with an ABFT capability is likely to have less sensitive bits in the logic level, it is yet to perform a systematic and quantitative study to investigate the impact of the ABFT on the logic circuit. Note that ABFT and AMFT are largely determined by the specific application, so it is hard to advance a concrete and consistent model for both forms of AFT. However, for AMFT, we propose a measurement-introduced fault model (M-FM), which systematically maps the measurement tolerance infor-

mation into logic level. This new fault model makes it necessary to re-study the existing logic-level fault analysis and fault-tolerant optimization techniques, since the sensitivity of each bit is redefined.

Two case studies are performed in the thesis to quantitatively analyze impacts of ABFT and AMFT on circuit reliability. Experimental results show that ABFT reduces the circuit's error rate by 18x for the matrix multiplication application. Further, using our new AMFT-based fault model M-FM, 3x improvement in circuit reliability is achieved for discrete convolution, which is a key operation in DSP applications.

1.3 Outline of thesis

The thesis is organized as follows. Chapter 2 introduces preliminaries. In Chapter 3 and 4, we respectively analyze two categories application-level fault tolerance, discuss their impacts on fault tolerance synthesis for FPGAs, and perform respective case study to qualitatively illustrate the importance of these two categories of AFT on circuit robustness. Finally, we conclude the thesis in Chapter 5.

Chapter 2

Preliminaries

2.1 Radiation Effects on Integrated Circuits

Fault tolerance for semiconductor devices is becoming a more and more important issue since upsets were first discovered several years ago in space applications. The research in various fault-tolerant techniques to keep the IC devices operational in a radioactive environment, such as space shuttle and satellite, has become a hot topic since then. The space shuttle, consisting of a number of various digital and analog IC devices, suffers from the sensitivity to radiation and thus must be protected for space operation. Single event effect (SEE) is the main concern in space,which has serious effect on space shuttle electronics and might lead to serious consequences for space tasks, like functional failure or information loss.

Ions or electro-magnetic radiation can cause a change of state of sensitive node in semiconduct devices, such as a microprocessor, memory, or power transistors. The state change is a direct result of the free charge caused by ionization in or close to an important node in a logic element. These unwanted changes or errors in device output or operation are known as soft errors, and are the most common type of single event upsets (SEUs) [2]. Its main consequences are bit flips in the memory unit. The SEU itself does not cause permanently damaging to the transistor's or circuit's functionality, i.e. non-destructive event, unlike single event latchup, or single event burnout. However, under certain circumstances, a parasitic thyristor inherent to CMOS designs can be activated and cause short-circuit from power to

ground, in which case it is referred to as latchup. It often causes destructive damage to the device from thermal runaway in the absence of proper countermeasures.

With the continuous technology evolution, IC devices with more and more categories of architectures, a large amount of embedded memories have been devised. Also, the fabrication process of semiconductor devices is in a rapid evolution in terms of transistor geometry, power supply, speed and logic density. However, the amazing device shrinking,operating speeds improvement and power consumption reduction have significantly narrowed the noise margins [15] [16]. The very submicron (VDSM) ICs are suffering from the increasingly various internal noises. Right now the fabrication process is approaching a point where it is impossible to manufacture ICs free from upset effects,which is mainly related to SEUs. The necessity to protect those ICs from upsets is becoming more and more important for an acceptable reliability [17].

2.1.1 SEU in Digital Logic Circuit

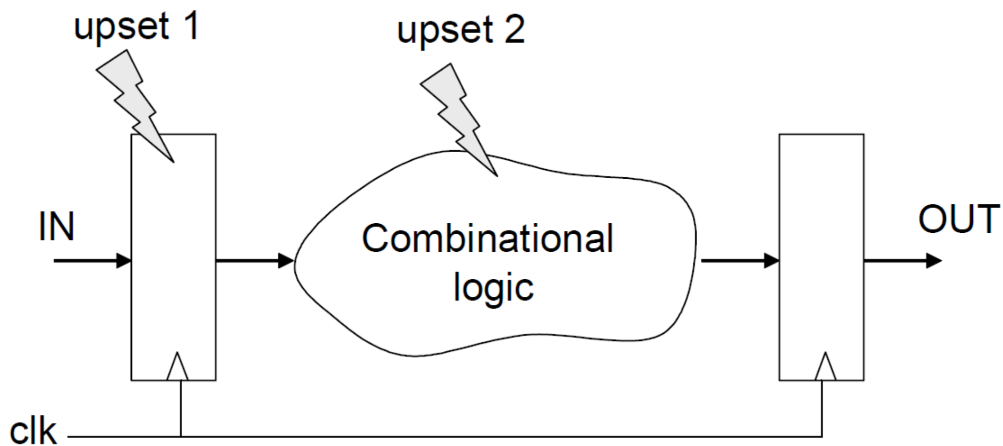


Figure 2.1: SEU in Digital Logic Circuit [3]

A single particle can hit either the combinational or sequential logic in the circuit as shown in Figure 2.1, which shows a typical topology for nearly all digital ICs [9] [10] . The input data latched in the first register is released on the rising or falling edge, which is fed to the following combinational logic for functional

operation at the same time. The output of combinational logic reached the second register and latched at the next rising edge or falling edge. All the date latching happens right on the active edge of the clock.

SEU in Sequential/Memory Logic

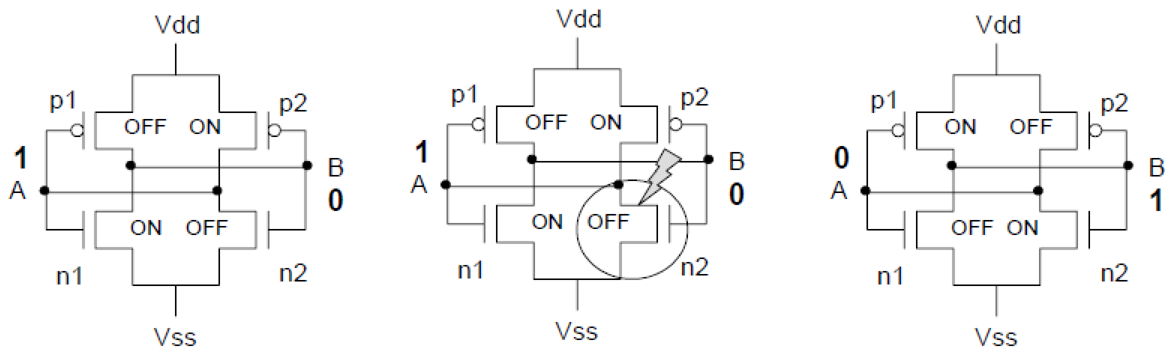


Figure 2.2: SEU in a SRAM Memory Cell [3]

When one of the sensitive nodes of the memory cells, like a drain in an off state transistor, is stroked by a charged particle, it will generate a transient current pulse, which might potentially turn on the gate of the opposite transistor. It can finally lead to an inversion, ie. a bit flip in the memory cell. As shown in Figure 2.2, an SRAM memory cell has two stable states, representing '1' and '0' respectively. When an energetic particle hits the transistor, it will cause the change of transistors' state and a bit-flip happens. This effect is exactly what we call SEU, one of the major concerns of digital circuits in radioactive environment.

SEU in Combinational Logic

When the combinational logic is hit by an energetic particle, it can generates a transient current pulse, which is called single transient effect (SET) [11]. If the combinational logic path is fast enough to propagate this induced transient current pulse, then it will appear at the input of the second register in Figure 2.1 and be latched as a valid signal.

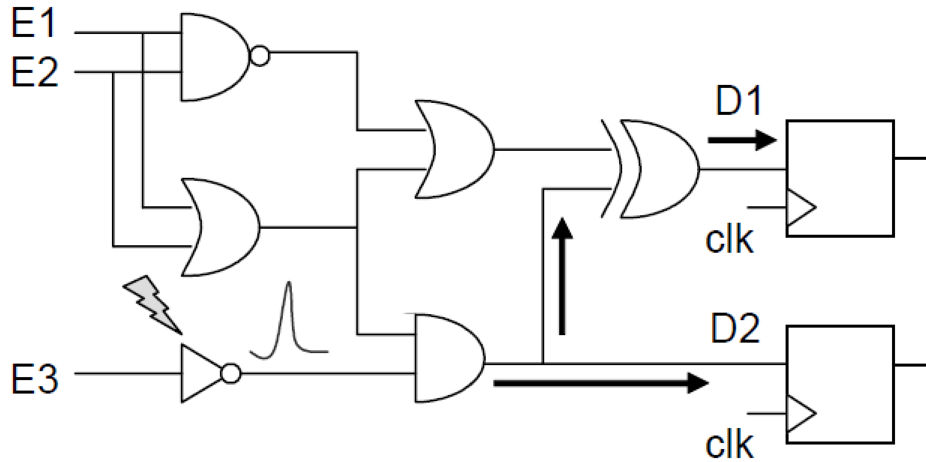


Figure 2.3: SET in a Combinational Logic Circuit [3]

Whether the SET will be stored by the latched as real data depends on the timing relationship between its arrival time and the active edge of the clock. Figure 2.3 shows the signal path in a combinational logic. The probability of a SET becoming a SEU is discussed in [12] and [13]. In a large circuit composed of many paths, the analysis of SET is very complex. Some methodologies like the classic timing analysis can be adopted to analyze the probability that a SEU in a combinational logic is stored by a memory cell or causes functional failure, as detailed in [14]. Also, the width of the induced transient current pulses can be measured to get a more precise mathematical model for fault-tolerant analysis. In addition, a single SET can produce multiple transient pulses at the output because of the logic fan-out. Therefore, once the SETs are captured by the flip-flops, they can cause multiple bit upsets (MBU).

2.1.2 SEU Classification

In terms of the number of upsets that occur simultaneously in the circuit, SEUs can be categorized into first, second and third order effects. A SEU is called a first order effect and multiple bit upsets (MBU) are classified into second or third order effects. MBU are caused by a single charged partition that travels through the IC at a shallow angle and strikes two sensitive junctions at the same time by ionization or

nuclear recoil [18]. Experiments have shown that multiple upsets can be provoked by a single ion in memory components under heavy ions fluxes [19].

There are three types of MBU. The first type is that a single charged particle hits two adjacent nodes located in two distinct memory elements, which is called second-order effect. This type MBU can be eliminated through proper placement. For example, the memory cells in the same register can be placed apart away from each other to avoid the simultaneous strike by a single particle that affects adjacent cells in a same data memory. The second type is caused by a single particle striking two nodes in the same memory cell, which is categorized as a third-order effect. To minimize the occurrence of this type of SEU, we can separate those critical nodes by large distances during the physical layout phrase in a circuit design. The third type of MBU is that multiple particles strike multiple nodes and thus provoke multiple upsets in memory cells. We can analyze this type of MBU by treating them like a group of SEU. According to [20], the majority of multiple upsets in adjacent memory cells are caused by a single particle. The probability of multiple charged particles in adjacent memory cells provoking multiple upsets in less than one second is quite low.

2.2 Introduction of FPGA

A field-programmable gate array (FPGA) is a semiconductor chip that can be programmable or reconfigurable after manufacturing using a bitstream of configurable file. The design configuration file is usually described using a hardware description language (HDL), such as Verilog HDL or VHDL. Compared to the traditional application specific integrated circuit (ASIC) with fixed logic function, we can implement any logic functions and update them if required even after shipping. The flexibility offered by FPGA allows us to program product features, adapt to new standards, and reconfigure the circuit for new applications after the product has been installed in field, thus called ”field-programmable”.

2.2.1 FPGA Architecture Overview

Inside an FPGA chip, there are lots of configurable logic blocks, and a network of programmable interconnects that connect the blocks to be wired together. The logic blocks can be configured to implement any complex combinational logic. In addition, there are some other programmable logic blocks, like memory elements, which can be simple flip-flops or more complete RAM blocks.

In the rest of this section, we will take the Virtex family FPGA from Xilinx, one of the most popular SRAM-based programmable devices used in the market nowadays, as an example to detail the principles of FPGA devices. Virtex FPGA has high density and high-performance, and supports a wide range of applications. Its fabrication process is on thin-epitaxial silicon wafers using 0.22 μ CMOS process with 5 metal layers [21]. As a VDSM design, it is very sensitive to the radiation and some fault-tolerance techniques must be posed for tasks in radioactive environment.

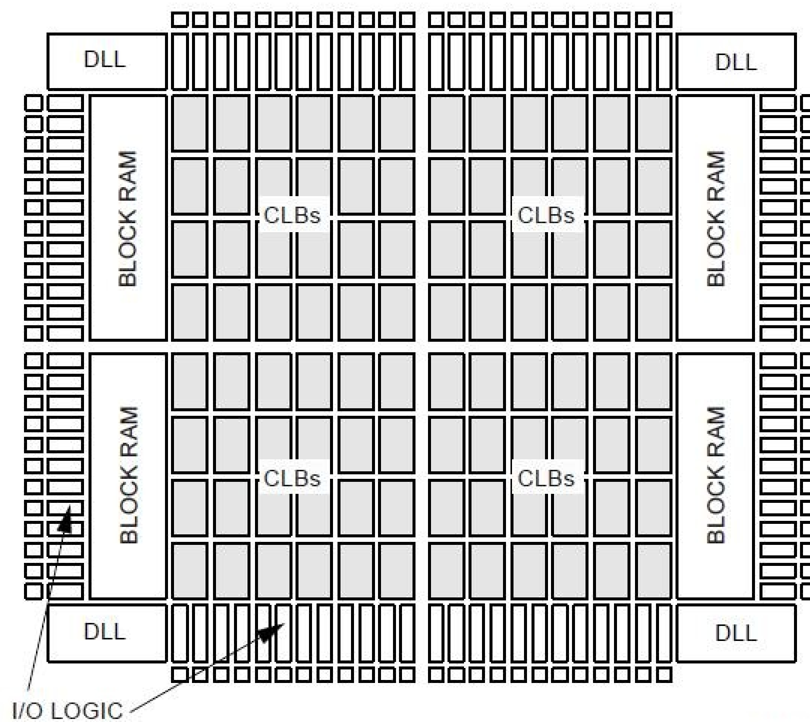


Figure 2.4: Xilinx Virtex FPGA Architecture Overview [22]

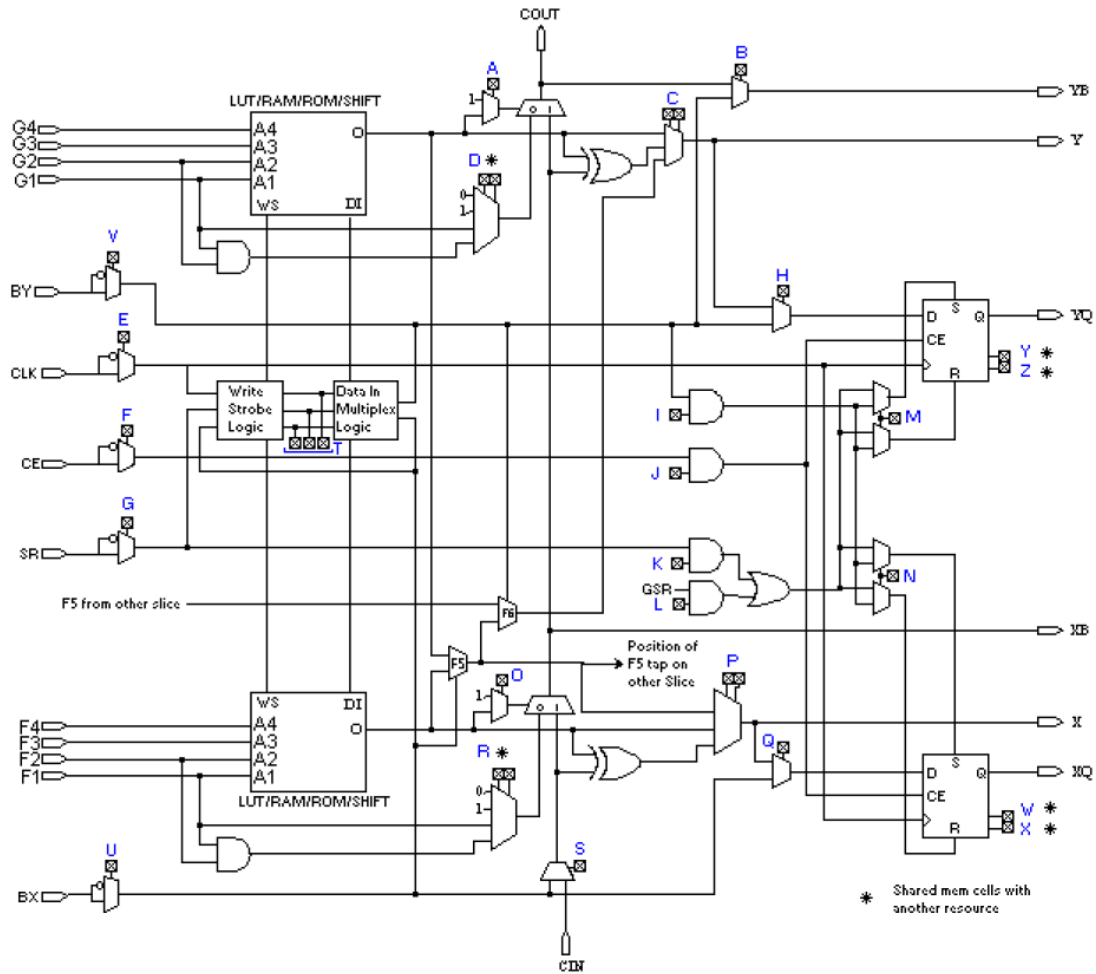


Figure 2.5: One Slice Structure [5]

Configurable Logic Blocks

Virtex FPGAs are consisted of a two-dimensional array of CLBs, a sounding ring of input/output blocks (IOBs), and some block RAMs (BRAMs)on the west and east edges. The delay-locked loop (DLL) units can be used to produce stable clocks from external clock sources. The CLB are the type of building blocks that contain programmable elements for implementing customizable logic, flip-flops and so on. Each CLB contains two slices and each slices is consisted by two 4-input look-up tables (LUTs). two flip-flops, carry logic and routing resources, as shown in Figure 2.5. The IOBs are used for communicating internal signals with external devices. BRAMs have the capacity of up to several MBs. In addition, there are

other embedded blocks on modern FPGAs to provide high performance, like DSP, floating-point model, etc.

Routing Resources

Those slices, I/O, clocking, BRAMs, and other resources are connected with each other through a large amount of programmable routing. As shown in Figure 2.6 the routing resources allows data to be fetched or received from other CLBs. The input muxes and output muxes are used to route signals to and from the slices.

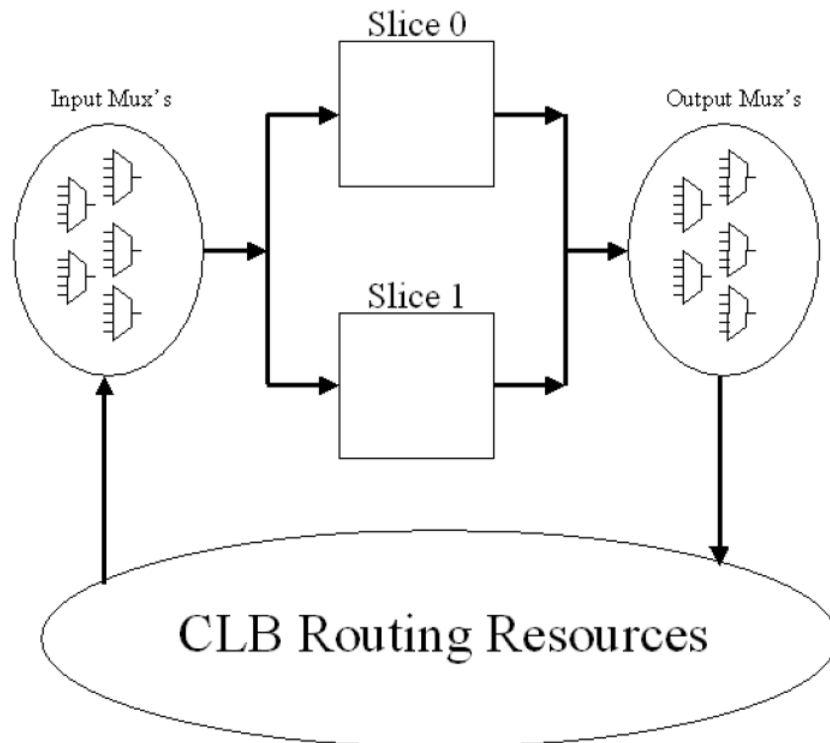


Figure 2.6: Routing inside FPGA [22]

Look Up Table

A K-input look-up table is essentially a memory that can store 2^k bits data. It can implement any boolean function of K variables by storing the truth table of that boolean function. Table 2.1 is the truth table for a boolean function $x = ab + \bar{b}c$. Simply mapping the truth table to the LUT memory, with a 8 to 1 multiplexer con-

Table 2.1: A 3-input Truth Table

a	b	c	$x = ab + \bar{b}c$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

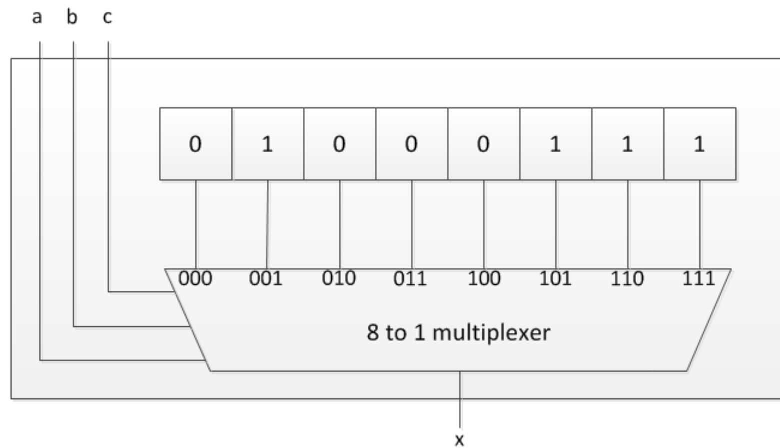


Figure 2.7: A 3-input Look-up Table

trolled by the variables a, b, c, we can implement the function in the LUT as shown in Figure 2.7. Combined with the flip-flops within each slice, we can implement any digital logic on FPGA.

The bitstream of a configuration file for an SRAM-based FPGA contains all the mapping and routing information, including the values held in LUTs and BRAMs, the interconnection between the slices, and the selection of the resources modes, like I/O standards, drive strengths, etc.

2.2.2 FPGA Design Flow

Typically, for an FPGA-based system design, the whole engineering process involves the following stages, as shown in Fig. 2.8.

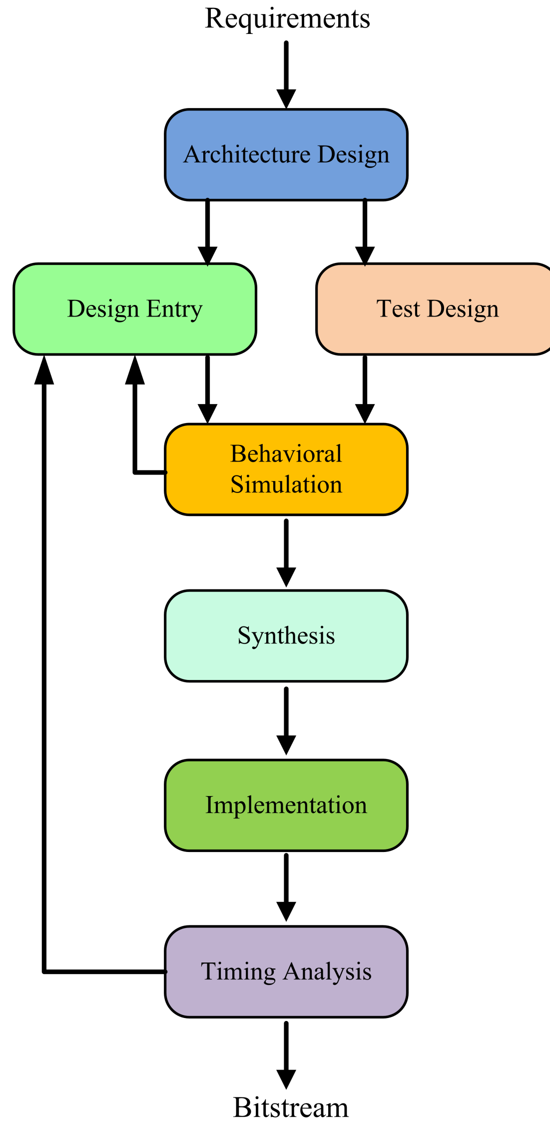


Figure 2.8: FPGA Design Flow

1. Architecture Design

   In this stage, we need to analyze the customer requirement and decompose the proposed problems and choose which type or architecture of FPGA is the best suitable. The output usually is a document describing the target device model, system architecture and structural blocks, their functions and interfaces.

2. Design Entry

   There are different techniques for design entry: hardware description lan-

guage (HDL), such as VHDL and Verilog, schematic based, or a combination of both. The HDL-based design is usually good at dealing with large and complex design, or implementing it in an algorithmic way. But the synthesis tools for HDL may not produce an optimal design and thus it may suffer from the performance and area consumption. Using schematic block to describe a design is more straightforward and simple, but hard to manage a big project.

3. Test Design

   In this stage, we need to write the testbench, which is used as a test environment to ensure the HDL design of the project is correct.

4. Behavioral Simulation

   Combining the design entry and testbench, we can conduct behavioral simulation. This stage aims to check the HDL correctness by comparing the output waveform or data of the HDL model and the behavioral model.

5. Synthesis

   The synthesis stage converts the VHDL or Verilog code into a device netlist format, which is basically a standard written digital circuit schematic with logic elements, like flip-flops, gates, etc. Synthesis tools will check code syntax and analyze the hierarchy of the design for optimization.

6. Implementation

   In this stage, the synthesizer-generated netlist is mapped into a particular device's programmable resource. There are three sub-stages involved: translate, map, place and route, which allocate FPGA resources, and place and route them to meet the design constraints like area, timing limitations. The file generated in this stage is a bitstream configuration file which can be downloaded into FPGA directly.

7. Timing Analysis

   In this stage, special tools would check whether the implemented design

meets the timing constraints, like the maximum clock frequency, critical path delay, clock skew etc.

2.3 SEU Effects on SRAM-based FPGAs

The evolution of fault-tolerant techniques has a close relationship with the target devices. To mitigate the SEU effects on a specific device, we need to conduct a detailed research on its architecture and upsets phenomena in order to select the most suitable SEU mitigation solution. Due to its advantage in terms of flexibility, low non-recurring engineering (NRE), and high performance, FPGAs are increasingly demanded in a lot of areas, like telecommunication, encryption, network routing and especially spacecraft tasks where the environment is extremely radioactive. The space applications using SRAM-based FPGAs with high density of logic resources, embedded processors and memories brings the necessity of exploring new SEU mitigation techniques for SRAM-based FPGA. In the rest of this section,we will demonstrate the inherent SEU modes on SRAM-based FPGAs.

Some SEU failure modes exist only in FPGAs rather than the conventional ICs. For example, a single bit upsets in the configuration file, the routing wire may connect to the two endpoints, and the fan-in may become larger unexpectedly and a LUT function can be corrupted. There are many ways to classify the SEU modes on SRAM-based FPGAs. In this thesis work, we follow the seven types of SEU modes given in [5]: mux select, PIP short, PIP open, buffer off, buffer on, LUT value change, and control bits change.

1. Mux Select

   Multiplexers resources are a large part of internal interconnection network. Almost all the circuit inputs and outputs needs multiplexing. They are very sensitive to SEUs. Any change in the select bits can cause a function failure because of the wrong routing configuration. Figure 2.9 is an example of mux select failure.

2. PIP On/Off Failure

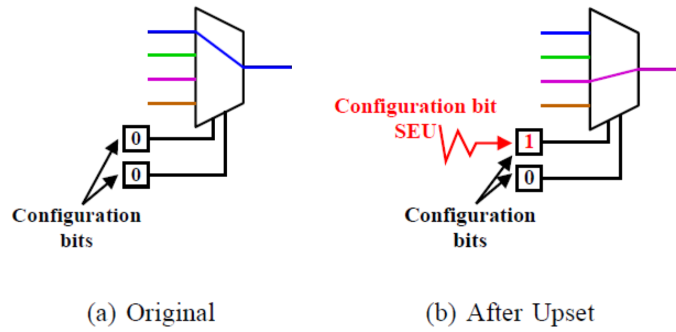


Figure 2.9: Mux Select Failure [5]

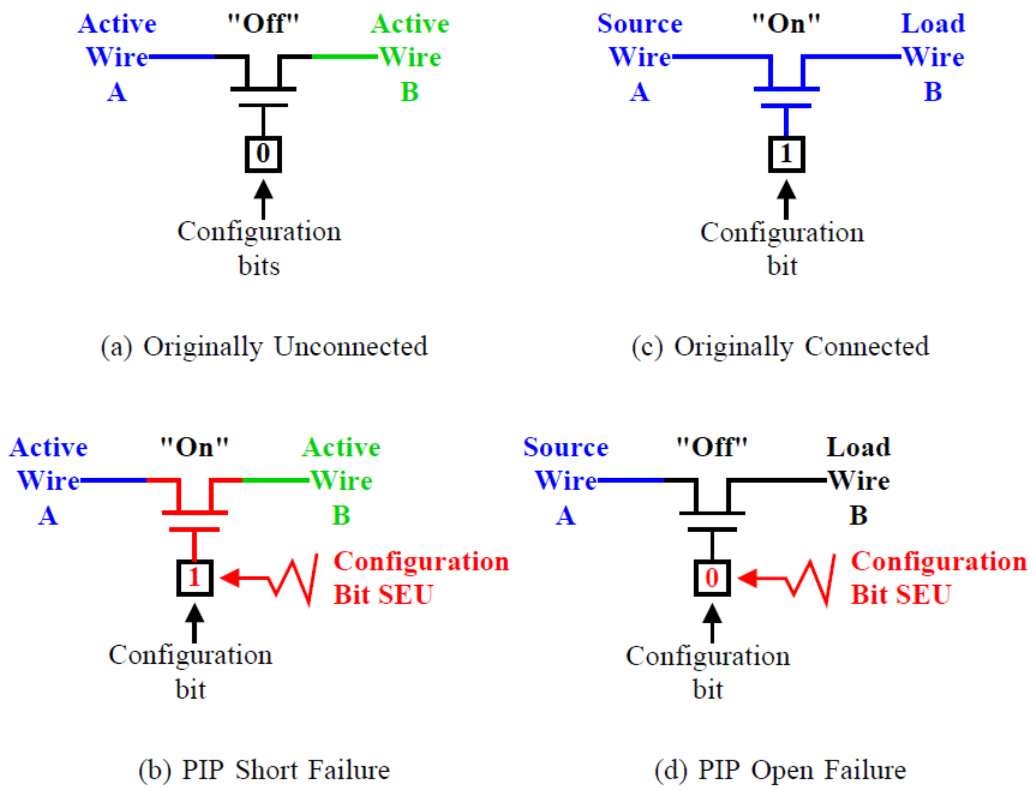


Figure 2.10: PIP Failure [5]

Programmable interconnect point (PIP) is another main source for FPGA routing network, which is a pass transistor between two wires that can be in state on or off,i.e.connected or unconnected. As shown in Figure 2.10(a) and (b), it is a PIP short failure. It will cause two different functions in the

circuit shorted together and may lead to output errors, contention and power burnout. In Figure 2.10(c) and (d), a PIP open failure happens. It can make the interconnection broken and thus stop the information flow from one module to another one.

3. Buffer On/Off Failure

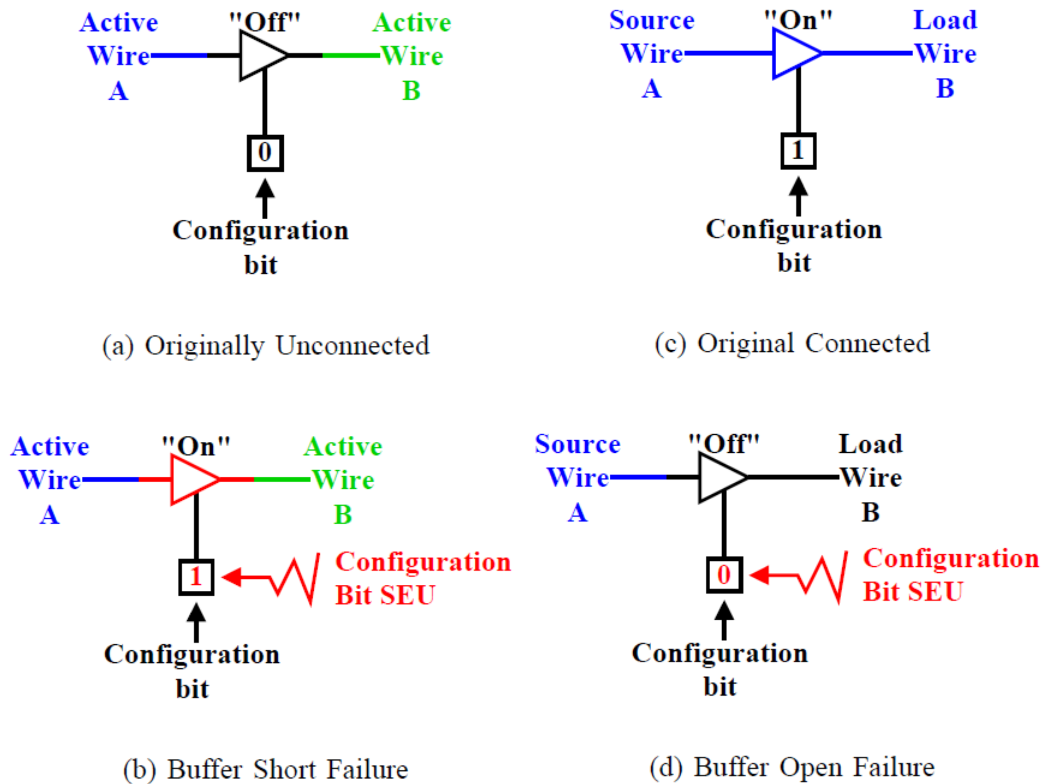


Figure 2.11: Buffer Failure [5]

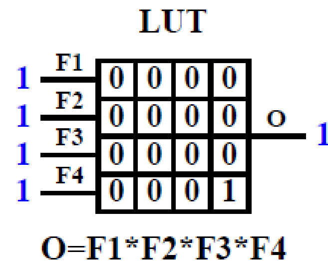
A buffer is also considered as routing resource, which works as a driver to improve the fan-out ability of the circuit, that can be configured as on or off. Similar to the PIP failures, buffer failures also have two modes as shown in Figure 2.11.

The difference between those two failures is that the PIP failures, caused by the malfunction of a pass transistor, could affect both sides circuits of the PIP node while as a buffer failure, it is usually placed on the outputs of

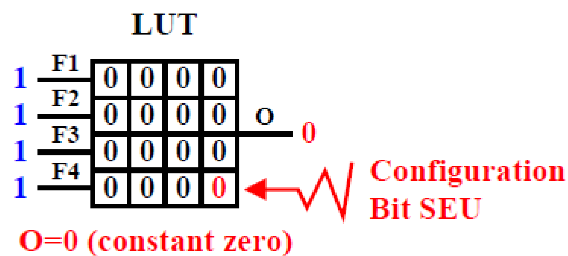
multiplexers, only the output side is affected.

4. LUT Value Change

LUT resources are the main programmable elements used by FPGA to im-



(a) Original 4-input AND Function



(b) Upset LUT Function (Constant "0")

Figure 2.12: LUT Value Change [5]

plement logic functions. A LUT value change may impact the function implemented, which may cause constant or temporary output errors. As shown in Figure 2.12, a AND function is implemented in a 4-input LUT. The single bit upset from 1 to 0 in the true condition would cause a constant-zero function. In this case, for most of the possible inputs, the output would still remain correct. This type of SEU causes LUT value change in FPGA is the main concern of this thesis.

5. Control Bits Change

Compared to the LUT failure, the control bits change has a much worse impact on the whole circuit. There are many control bits within a slice on

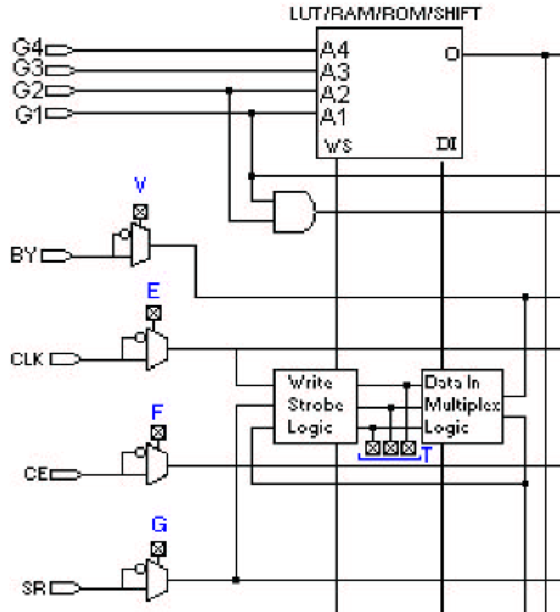


Figure 2.13: Control Bits Change [5]

Virtex FPGAs. As shown in Figure 2.13, Bits V, E, F, G, T are all control bits. V, E, F, G are programmable inversion bits. A change on those bits may lead the value on that wire inverted and thus may cause a functional failure if it is a sensitive wire. The T bits control the operating mode of LUT, working as a normal LUT, shift register, or part of a distributed RAM. If a LUT is being used as a normal LUT, the change may cause it into a role of shift register, and thus the function may fail.

Research in [4] and [5] show that LUT resources are most sensitive to SEUs and routing resources are the most critical resources and have the most significant impact on the configuration bitstream SEUs.

2.4 SEU Mitigation Techniques for SRAM-based F-PGAs

The first SEU mitigation technique used in spacecraft for many years was shielding, which can keep the particle flux at a low level instead of eliminating it complete-ly. However, due to the evolution of fabrication process technology, the integrated

circuits are becoming more and more sensitive to the radioactive particles, where the conventional shielding solution does not work anymore. Consequently, new mitigation techniques need to be applied to mitigate the radiation effect.

In this section, the new SEU mitigation techniques for SRAM-based FPGA will be investigated in terms of their developing cost, reliability, time-to-market, and power consumption etc.

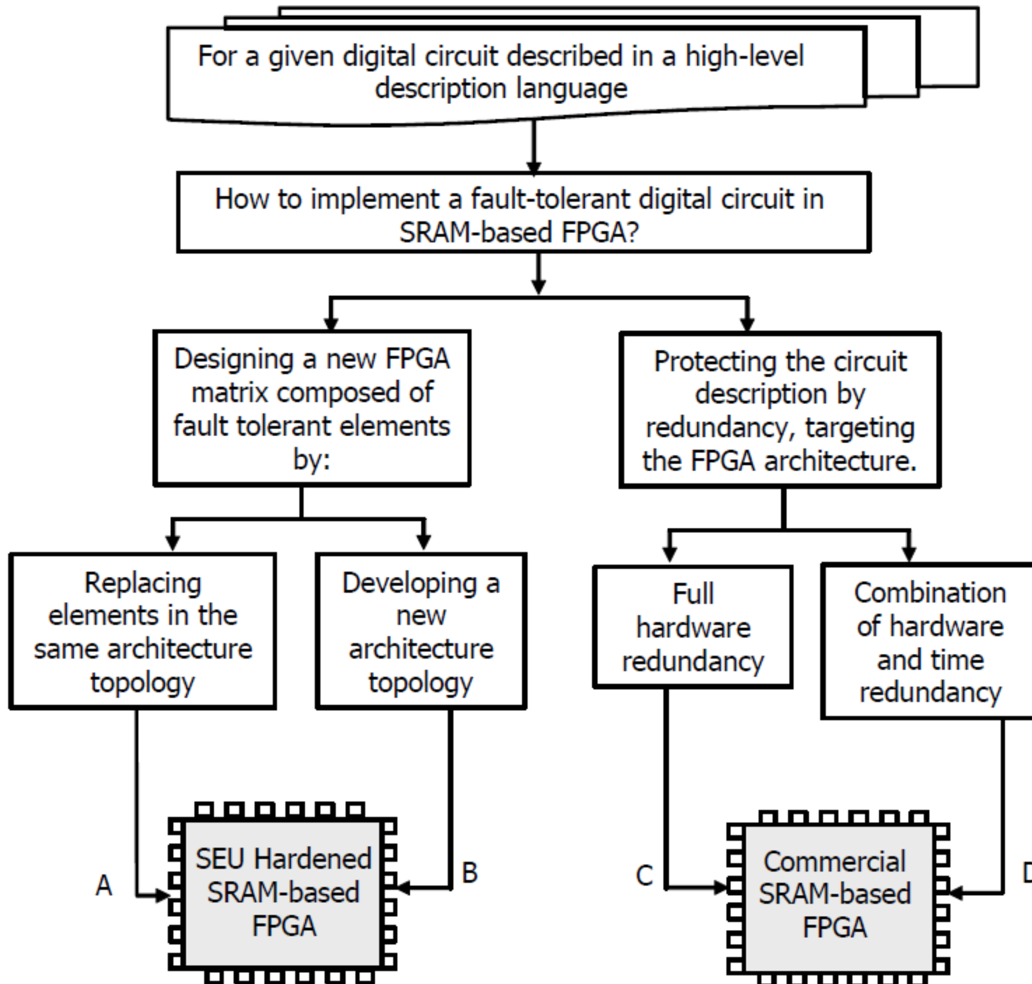


Figure 2.14: Fault-tolerant Design for SRAM-based FPGA [3]

As shown in Figure 2.14, for a given digital circuits descried in a high-level hardware language, there are two ways to implement fault-tolerant design in SRAM-based FPGAs in general.

The first way, from the perspective of a new architecture or design of the pro-

grammable resources, is to design a new FPGA array composed of fault-tolerant elements. Those new elements designed for fault-tolerance can replace the old ones in the traditional architectural topology or in a new fault-tolerant topology. The developing cost for such new devices are expensive because this process involves fabrication technology, long IC developing cycle. Another way is to mitigate SEUs at a high-level rather than the chips itself using different kind of redundancy in the circuit, in which case we can use a commercial off-the-shelf (COTS) FPGAs to implement the SEU mitigation solutions applied to the design description before synthesizing the design. This approach eliminates the requirement of new chip architecture development and fabrication and reduce the cost. The users need to choose the optimal way to protect their design and thus have the flexibility of trading off the overheads in terms of performance, area, and power consumption.

In Figure 2.14, there are four different ways of implementing a fault-tolerant FPGA design with different costs. The cost of B solution is higher than A, both of which are much higher than C, which is also higher than D. Each of them has its appropriate application field since they pose different constraints. Since the high-cost fabrication and long time-to-market, the high-level solution C and D tends to be more attractive. In following subsections, we will discuss both SEU mitigation techniques.

2.4.1 Architectural Mitigation

In [24], the author has developed a FPGA for space and military applications using the following SEU mitigation technology: a radiation hardened nonvolatile EEPROM transistors, SEU immune storage circuits for SRAM implementations, new FPGA architecture specially designed for radiation-hardened circuit design.

Actel also developed an SRAM-based FPGA specially, in which case the standard SRAM memory cells are replaced by resistor-decoupling memory cells, as shown in Figure 2.15. This special memory can avoid upsets by inserting resistors in the feedback path which works as a filter to the transient pulse provoked by the charged particles. This FPGA also adopt another mitigation principle: storing the

same data in two different locations, called DICE cells [25]. The DICE cell is able to avoid upsets by storing the data in distinct parts, where if one part is damaged, the other one is isolated by the cell architecture and remains well. In [26], the author concludes that the effectiveness of both approaches are limited in the circumstances of multiple bits upsets (MBU). And the redundancy of DICE memory is less effective than the resistor solution by two orders of magnitude in terms of upset rates.

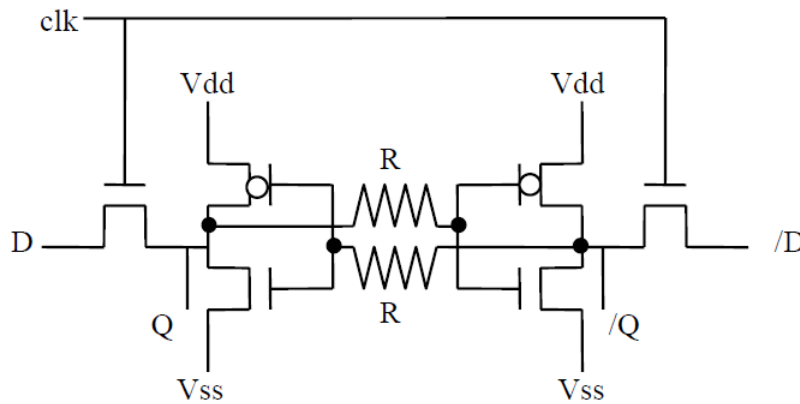


Figure 2.15: Resistor Hardened Memory Cell [3]

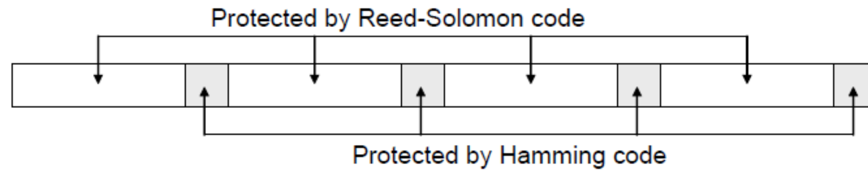


Figure 2.16: A Memory Row protected by Hamming and RS code [3]

A new approach has been developed for multiple bit upsets in VDSM memories [28], which combines hamming code and Reed-Solomon code with single symbol correction capability. It can achieve 100% correction of double fault correction with low cost RS code. Hamming code protects bits between RS symbols. Figure 2.16 shows the insertion of Hamming and RS code protection in a memory row. The final architecture of double error tolerant memory is shown in Figure 2.17.

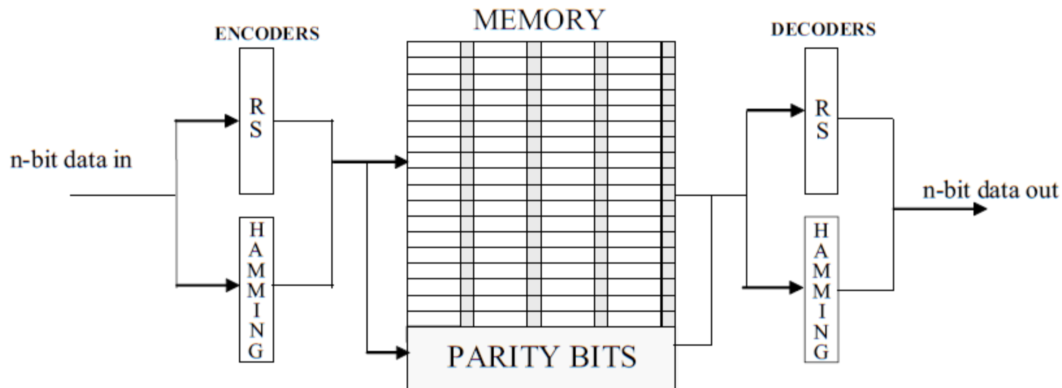


Figure 2.17: Hamming and RS Code Protected Memory Architecture [3]

2.4.2 High Level Mitigation

The architectural SEU mitigation solutions mentioned in the previous section can achieve a high level of reliability. However, the inherent cost of architectural solutions is very high, because they change the matrix,adopts expensive fabrication process and long developing cycle. Very few FPGA vendors are investing money in designing FPGAs that is specially adopted in space and military applications. A less expensive way for fault tolerant FPGAs is to protect the design using high level SEUs mitigation techniques.

Xilinx Virtex family protects the design by combining triple modular redundancy (TMR) and scrubbing [29].

Triple Modular Redundancy

Triple modular redundancy is a common methodology to improve system reliability by means of introducing hardware redundancy. As is shown in Figure 2.18, each working component is replicated three times, and a voter is used to compare their output signals and select the majority value. If any single component is faulty, the error will not be propagated to the next stage, since the other two correct values have decided the value of the voter.

Since the design on a Virtex FPGA is mapped into various structure types: I/O logic, state-machine logic, Block RAM, DLL etc. We can apply TMR into those

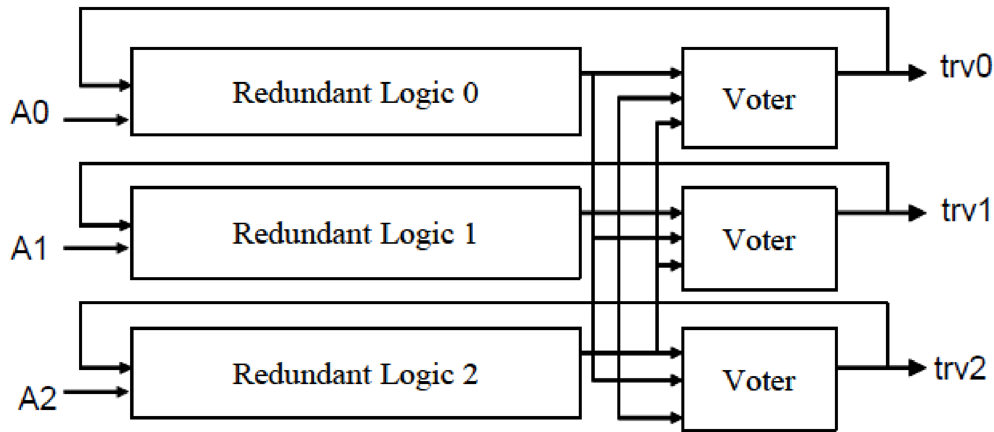


Figure 2.18: Triple Modular Redundancy [3]

logics appropriately to ensure reliability. For example, we can triple the Block RAM and I/O logics and vote for the majority so we can have the right output when a single module corrupts.

Although TMR offers tremendous improvement in reliability, its large overhead on area and power still limits its applications only to extreme critical missions. As a trade off between reliability and system overhead, partial TMR [44] technique has been proposed as an intermediate choice. By only replicating a small portion of error critical components, it greatly reduces the system overhead.

Scrubbing

As upsets accumulate over time in the matrix, the TMR technique is not sufficient to ensure the reliability for a long time. Since the upsets in the LUTs and routing cells will stay until the next reconfiguration of the device. There it is necessary to perform a scrubbing action. i.e. reconfiguration at a certain frequency so we can clear up the accumulated errors to ensure the circuit reliability.

The first technique of cleaning up upsets was reading back the configuration bitstream of the matrix and reloading the original one if any errors are detected [30]. The problem with this approach is that it is too time consuming.

A better way for SEU correction is simply reload the entire CLB segments at a chosen frequency without readback and detection of errors [31], called ”scrub-

bing". It allows the system self-repair SUEs in the configuration memory without interrupting its operations in a relatively short time. As shown in Figure 2.19, the scrubbing action is done through the Virtex selectMap interface. In this model, an external oscillator generates configuration clock driving the PROM and FPGA. The new data is available in each clock cycle on the PROM data pins, XQR18V04 in Figure 2.19. The time for scrubbing depends on the generated configuration clock. We can decide the scrubbing rate based on the expected upset rates for the given application.

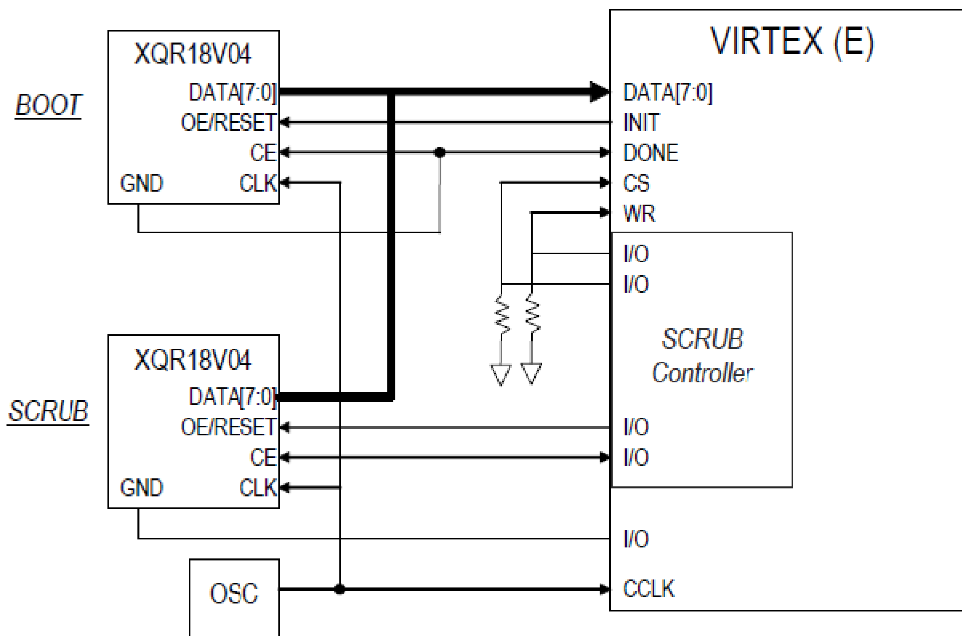


Figure 2.19: Xilinx Virtex Scrubbing Scheme [3]

2.5 Logic-Level Fault Model

The logic-level fault model (L-FM) evaluates the sensitivities of different circuit components (e.g., SRAM bits, LUTs) to SEU induced soft errors. We focus on single fault here because it has been shown that simultaneous multiple SRAM bit flips seldom occur in real situation [7].

The criticality of SRAM bit b, $Crit_b$, is formally defined as the probability that an error (i.e., the value is different from the error-free reference circuit) can be

observed in circuit's primary outputs over all possible inputs, if b is flipped. As is shown in equation (2.1), we test all input combinations and check if an error occurs at primary outputs. However, due to the large number of inputs in real circuits, it is often impossible to enumerate all combinations. Instead, we perform Monte-Carlo simulation in which a large set of randomly generated input vectors are tested to get an approximation of the accurate criticality.

$$Crit_b = \frac{1}{2^n}|\{x|C(b)(x) \neq C(\bar{b})(x)\}| \tag{2.1}$$

Here $C(b)(x)$ represents the golden output, while $C(\bar{b})(x)$ means the circuit's output when the researched bit is flipped. n represents the number of inputs.

Accordingly, we define the error rate of LUT and entire circuit as the average criticality of SRAM bits contained respectively.

$$Error_rate(LUT) = \frac{1}{N}\sum_{b \in LUT} Crit_b \tag{2.2}$$

$$Error_rate(Circuit) = \frac{1}{M}\sum_{b \in Circuit} Crit_b \tag{2.3}$$

where N and M are the numbers of SRAM bits contained in LUT and circuit respectively. Practically, the criticality or error rate measures the severeness of that specific component malfunctioning for circuit functionality.

2.6 LUT-based Boolean Network

In this section, we first introduce the way to represent boolean function. Then, a system tool for synthesis and verification we applied to our own simulator will be discussed.

2.6.1 Boolean functions and representations

To represent and manipulate the circuit we hope to synthesis as well as to do efficient boolean reasoning, we need to properly represent boolean functions. There are many ways with different levels to represent boolean function, such as truth table, Sum of Products(Disjunctive Normal Form), Product of Sums(conjunctive Normal Form), Brinary Decision Diagram(BDD), Boolean Network and so on. Within

above, boolean Network representation is an efficient way we apply to construct logic reasoning engine. A Boolean network can be viewed as a directed graph $G(V, E)$ where V is the set of nodes abstracted from standard cells like LUTs and E is the set of directed edges connecting these nodes. Each node e is assigned a boolean function $f(e)$ which reflect the relationship of inputs and output of the node. The nodes in the lowest level are circuit inputs which include primary inputs(PIs) and other registers inputs, while the nodes in the highest level are circuit outputs which include primary outputs(POs) and other register outputs.

The fanin $FI(v)$ of a node v are all predecessor vertices of e, which can be defined as: $FI(v) = \{v'|(v', v) \in E\}$. The transitive fianin $FI_T(v)$ is defined as follows: if $a \in FI(b)$ and $b \in FI(c)$ then $a \in FI_T(c)$. The fanout $FO(v)$ of a node v are all successor vertices of e, which can be defined as: $FO(v) = \{v'|(v, v') \in E\}$. The transitive fianout $FO_T(v)$ is defined similarly. The fanin(resp. fanout) cone of node v is a sub-network whose nodes can reach the fain edges of n(resp. can be reached from the fanout edges of n) [38], which includes the transitive fanin(fanout) of node and node itself. These definition can be illustrated in Figure 2.20. For exampl, $FI(6) = \{4\}$, $FO(6) = \{7\}$, $FI_T(6) = \{0, 2\}$, $FO_T(6) = \{8\}$, and fanin cone of 6 node is $\{0, 2, 4, 6\}$.

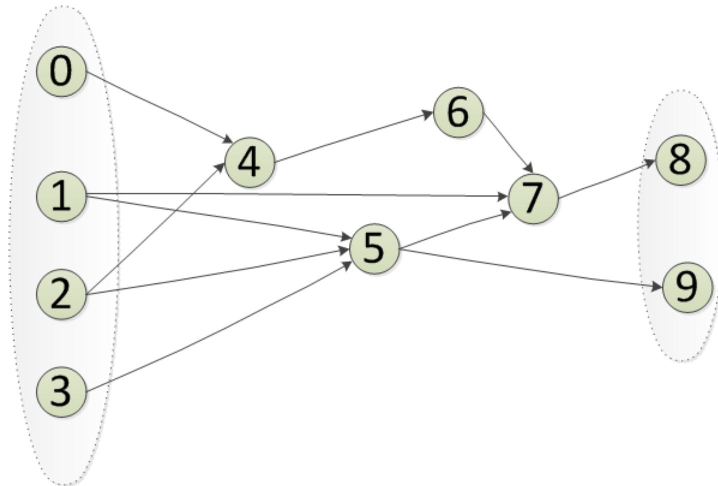


Figure 2.20: Illustration definitions of PIs/POs, fanins/fanouts and Transitive fanin/fanout cone

2.6.2 ABC: A System for Sequential Synthesis and Verification

ABC [41] is an open software system designed by Berkeley Logic Synthesis and Verification Group. As is shown in Figure 2.21, ABC provides a flexible programming environment to help verify and optimize the logic synthesis phase and technology mapping phase of design flow. Compared with traditional verification tool such as SIS, VIS, MVSIS, ABC will keep data structures simple and flexible just as the name suggested. A bunch of APIs provided can be applied to develop a wide range of applications.

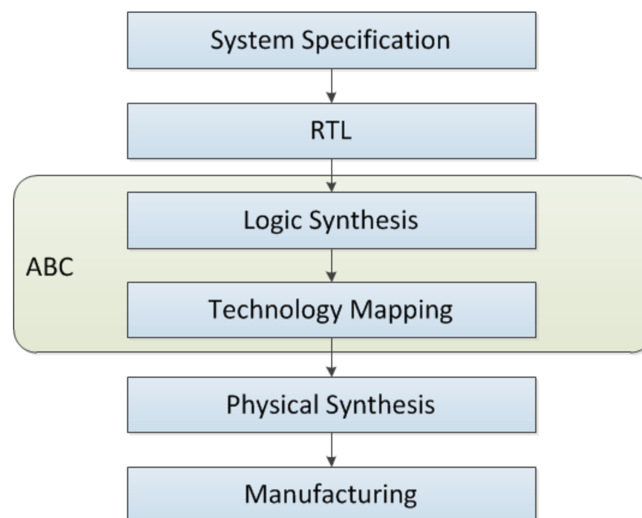


Figure 2.21: ABC Territory

ABC processes the design to create current network initially by reading specification, for example, from BLIF file to create the current network which is stored in memory during the running time. The current network can be further transformed step by step by specific system commands or APIs and can be finally written out for use.

Traditional netlist is supported by ABC, which includes net, logic nodes, latches, and PI/PO terminals. A logic network is another representation in ABC, which is essentially a netlist in which the nets are removed. And-Inverter Graph(AIG) is the specialized internal representation in ABC, in which each node is a two-input AND gate and each fanout or fanin has an optional attribute indicating inverter on

that edge. No matter what representation ABC provides, the data structure of internal component of networks is named as *object*. An object has a data field to indicate whether it is a net, a logic node, a latch or a PI/PO with unique ID to identify.

In a logic network, a node has a completely specified boolean function to correspond, which can be represented as an SOP, or a gate from standard cell library. When the function of node is represented using SOP, it will be as simple as it appears in a BLIF file, that is to say, in the form of truth table. For example, a two-input AND gate can be represented as ”11 1\n”, while a two-input XOR gate is ”01 1\n10 1\n” in C string format.

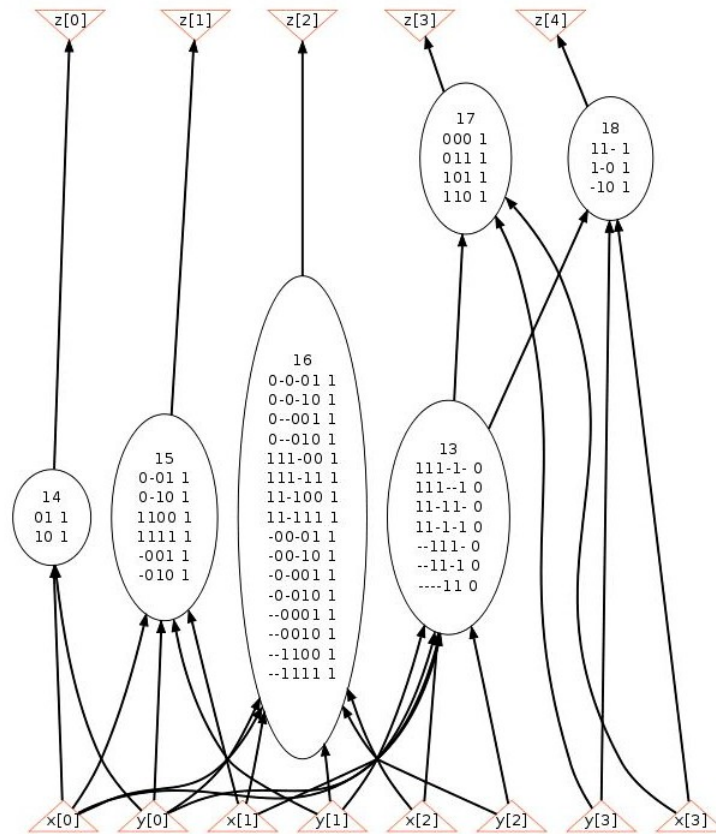


Figure 2.22: 4-bit Ripple Adder Network structure visualized by ABC

In a logic network, various iterators are available to iterate over different objects according to the requirement. For example, iterators over the fanins/fanouts of a node are *Abc_ObjForEachFanin* and *Abc_ObjForEachFanout*. Iterators *Abc_NtkForEachPi* and *Abc_NtkForEachPo* iterate over PI/PO terminals, While *Abc_NtkForEa*

chNode does not iterate over the terminals.

With the APIs and models supported by ABC, we can write our own fault simulator to conduct further experiments. The design is written in Verilog HDL and further converted into BLIF file. Our simulator with ABC API can read in this BLIF file to further converted into LUT-based boolean network as a directed acyclic graph(DAG) whose nodes correspond to LUTs and whose edges can be viewed as the relationship of connecting of two LUTs.

The logic network of benchmark of 4-bit ripple adder is shown in Figure 2.22. We can see 8 bits input and 5 bits output as PIs and POs. The middle nodes are represented by their truth tables and unique ID numbers. For example, No.17 node can be viewed as 3-input LUT whose stored value is corresponded to "01101001"(or "69"in hexadecimal) because"000","011", "101", and "110"correspond to the 0,3,5,6 bit in 3-input LUT value list.

Chapter 3

Algorithm-based Fault Tolerance

As mention in introduction, we find that there exist higher-level flexibilities named as AFT to compare with logic level fault tolerance in FPGAs. According to how to utilize the property of application to enhance fault tolerance, we divide AFT into two categories, ABFT and AMFT as shown in Figure 1.1.

In this section, we analyze the characteristics of ABFT. To further explore it-s impact on circuit reliability, we study the linkage between this flexibility and logic-level fault tolerance. A case study about matrix multiplier with checksum is discussed after that.

3.1 Characteristics of ABFT

ABFT reveals the inherent fault tolerance ability in the designated algorithm. The main source of this flexibility comes from the redundancy in the algorithm, which is used to prevent internal errors from propagating to primary outputs. The Conway's Game of Life [37] is a classic cellular automaton to simulate the life evolution process. At each iteration, each cell decides its next state based on a few rules. One of the rules is stated as:

- Any live cell with more than three live neighbors dies.

Because each cell has eight neighboring cells in two-dimension Conway's Game of Life, the error in accumulating the number of live cells (e.g., from 3 to 5) has a good chance to be masked and thus does not affect the evolution process. Actually,

such tolerance is common for a wide variety of applications related to behavior simulation, as human beings by their nature tend to make decisions based on (vague) ranges rather than a definite single value.

A fault-tolerant algorithm is sometimes distinguished by three major characteristics: encoding of the data used by the algorithm, redesign of the algorithm to operate on encoding, and the ability to recover the data in case of an error. One example is the fault-tolerant matrix multiplication algorithm [39]. As data encoding, a redundant row and column are inserted to the original matrix as checksums. The original matrix multiplication is redesigned to make use of that row and column, such that any single error in the value of matrix's element can be detected and corrected.

3.2 Impact on Circuit Reliability

Situating at a higher level, ABFT has a significant impact on circuit robustness. As an application must be finally implemented into a logic-level circuit, we need to study how this transformation will affect the circuit reliability.

ABFT aims at reducing fault rate by introducing extra redundancy in the target algorithm such that a fault can be eliminated in data flow. Intuitively, it should have a large impact on final system reliability. But we must note that such auxiliary components as error checking and correction units potentially consume hardware resource as well. Therefore, to make a fair evaluation on the gain of a fault-tolerant algorithm, we must compare the overhead with traditional technique like TMR.

The synthesis process shall be also aware of the algorithm-level fault tolerance to pass the fault masking ability into final implementation. In logic synthesis, if no further information is provided, the optimization engine tends to mix different parts to reduce area, through which the error masking ability at algorithm-level design can be lost. For example, after logic optimization the SEU can cause multiple computing errors in the matrix, which surpass the algorithm's ability to correct any single error [39]. To ease these unexpected side effects, we must feed ABFT information into synthesis engine. For example, in logic synthesis we must quanti-

tatively understand which parts are used for error handling such that they can not be multiplexed. In section 3.3, we will perform a quantitative study to analyze these factors.

3.3 Case Study of ABFT: Matrix Multiplier

Matrix Multiplication is one of the key operations in numerous applications such as signal and image processing. In addition, since it is composed of a large number of scalar addition/multiplication operations, it becomes a representative application for arithmetic circuits. In this section, we made a quantitive evaluation on the fault-tolerant matrix multiplier [39] to study the impact of ABFT on circuit reliability. Starting from verilog HDL implementation with error checking and correction algorithm, we transform verilog description into BLIF format using Altera's QUIP toolkit [45] to further evaluate reliabilities of matrix multiplier. To maximally utilize the parallelism of FPGAs, we unfold all arithmetic operations such that the computation can be finished in one clock cycle. Finally, we compare the reliability improvement.

The algorithm here we apply is closely related to the paper [39]. The column checksum matrix of matrix A is represented as $A^{\Box}$, the row checksum matrix of matrix B is denoted as $B^{\diamond}$, and the full checksum matrix of matrix C is expressed as C^{*}. These definitions can be formulated as below:

$$A^{\Box} = \begin{pmatrix} A \\ e^T A \end{pmatrix} \qquad B^{\diamond} = (B \ Be) \qquad C^{*} = \begin{pmatrix} C & Ce \\ e^T C & e^T Ce \end{pmatrix}$$

Then we can prove that the result of a column checksum matrix($A^{\Box}$) multiplied by a row checksum matrix($B^{\diamond}$) is a full checksum matrix(C^{*}) [39]. The corresponding matrix A, B, and C have the following relationship: $A * B = C$, Proof:

$$A^{\Box} * B^{\diamond} = \begin{pmatrix} A \\ e^T A \end{pmatrix} * (B \ Be) = \begin{pmatrix} AB & ABe \\ e^T AB & e^T ABe \end{pmatrix} = \begin{pmatrix} C & Ce \\ e^T C & e^T Ce \end{pmatrix} = C^{*}$$

With this property, we can detect whether there happens error when computing matrix multiplication and further locate the error and correct the error. For example,

given matrix $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ and matrix $B = \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix}$, we can compute the column checksum matrix $A^{\square}$ and row checksum matrix $B^{\diamond}$ first. $A^{\square} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 4 & 6 \end{pmatrix}$, $B^{\diamond} = \begin{pmatrix} 2 & 1 & 3 \\ 1 & 1 & 2 \end{pmatrix}$. Matrix multiplication conducts as usual without any SEU happens like this:

$$A^{\square} * B^{\diamond} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 4 & 6 \end{pmatrix} * \begin{pmatrix} 2 & 1 & 3 \\ 1 & 1 & 2 \end{pmatrix} = \begin{pmatrix} 4 & 3 & 7 \\ 10 & 7 & 17 \\ 14 & 10 & 24 \end{pmatrix} = C^{*}$$

However, if one SEU causes one element of matrix C, for example, the first row and the second column element 3 changes to let's say 5. Then the multiplier will show $C^{*'} = \begin{pmatrix} 4 & \mathbf{5} & 7 \\ 10 & 7 & 17 \\ 14 & 10 & 24 \end{pmatrix}$ as output. Error can be detected simply by checking the property of whether $C^{*'}$ is a full checksum matrix. Further by comparing the sum of C with its checksum, the inconsistence of the row and column will indicate which element is fault. Here the first row checksum 7 is not equal to 4 plus 5, so we can make sure the error happens in the first row computation. Then we can lock the error is in the second column when we find that the second column checksum 10 is not consistent with 5 plus 7. The correction of error also is handy through adding the difference between row checksum and sum of inconsistent row: $5 + (7 - (4 + 5)) = 3$.

Our case study applied on FPGA multiplying two 4-by-4 matrixes is conducted as Figure 3.1 illustrated. By using an extra row and column to encode the checksum of two input matrixes, the product matrix becomes 5-by-5 scale and the summation of row and column elements is contained in the redundant column and row. To check if there is an error in result, we simply add up left or upper four elements in one row or column and test whether it agrees with the checksum of that row or column. In the case of any single error in product matrix, we'll exactly detect inconsistency in one row and one column. The cross position of that row and column is the erroneous element. To correct the error, we can add the difference between corresponding column's (or row's) checksum element and the computed sum of

four other elements in this column (or row) just as the example given before.

Table 3.1 compares the error rate of this fault tolerant matrix multiplier with that of a normal 4-by-4 matrix multiplier. Not surprisingly, the error rate is reduced by 18x due to ABFT. We should note that though it is quite small, the non-zero fault rate indicates that the fault-tolerant matrix multiplier is still vulnerable to certain errors. The reason behind is that logic optimization tends to blur the boundary of different computing components and certain logic functions will be multiplexed. Thus, flipping of one SRAM-bit can affect the computing result in different parts and therefore go beyond the fault correction ability at algorithm level. This result further points out the motivation to take algorithm-level flexibility into consideration during logic synthesis.

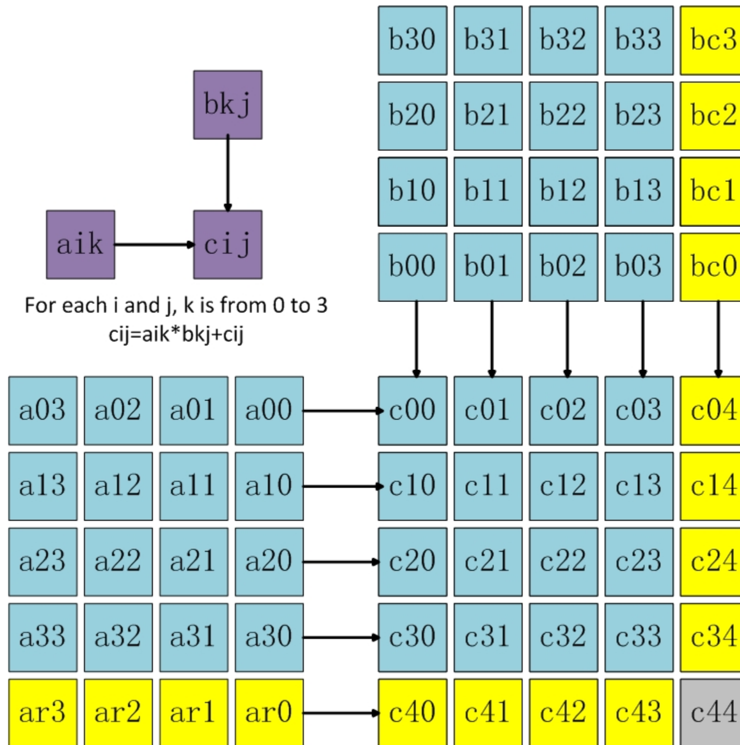


Figure 3.1: Theory of Fault Tolerant Matrix Multiplier

Another observation is that the 2x increase in area (i.e., number of LUTs) overhead is quite heavy. There're three major reasons leading to such large area, which can be explained using Figure 3.2. First, the multiplication scale is increased from 4-by-4 to 5-by-5, which leads to 9 more elements to be computed and each of which

Table 3.1: Comparison between Fault-tolerant and Normal Matrix Multiplier

Type	Error Rate	Ratio	# LUTs	Ratio
Fault Tolerant	0.0012	1x	3580	1x
Normal	0.02201	18x	1626	0.45x

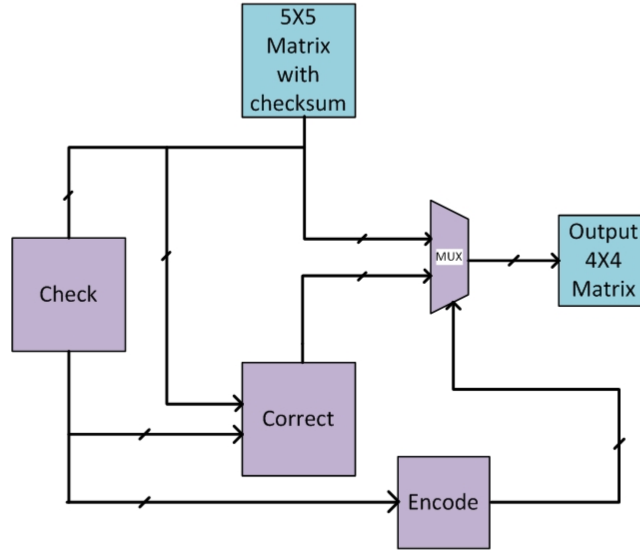


Figure 3.2: Error Detection and Correction

needs a few multiplications and additions. Secondly, the error detection and correction module incur further resource cost. Thirdly, every output needs a large MUX to select from the original or error corrected values.

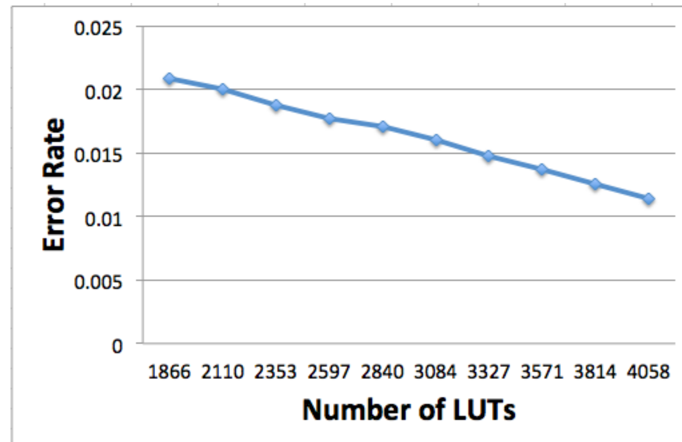


Figure 3.3: Criticality Reduction using Partial TMR for Normal Matrix Multiplier

To fairly evaluate the gain of utilizing algorithm-based fault tolerance, we com-

pare it with normal matrix multiplier with partial TMR optimization. Figure 3.3 shows the trend of error rate reduction with different TMR percentage. To achieve the same error rate level, normal matrix multiplier needs roughly 4000 LUTs, which is 12% larger than the fault tolerant one. As such, it is obvious beneficial to consider high level fault tolerance such as ABFT.

Chapter 4

Algorithm Mapping Fault Tolerance

AMFT is also application-dependent. So in this section, we discuss one specific case of AMFT which maps measurement property into logic level, and propose a corresponding measurement-introduced fault model (M-FM) to modify the logic-level fault model (L-FM). Impact of AMFT on circuit reliability is analyzed in detial and is verified in case study of discrete convolution.

4.1 Measurement Introduced Fault Model (M-FM)

One kind of algorithm property utilized by AMFT mainly comes from the specific interpretation of circuit's output vectors based on different applications and working environment. Typically, in cyber-physical systems, DAC is used as the interface between DSP and the outside world. Due to technology limitations, the DAC often has a precision boundary. As a result, even if the error is observed at DSP's outputs, it still does not matter whenever the value is within that precision boundary.

This is a big difference from L-FM introduced in 2.5, which has zero error tolerance since it does not take AMFT into consideration. Based on that observation of the existence of such application tolerable output deviations, we propose the measurement-introduced fault model (M-FM). That is, even errors can be measured at primary outputs, they can be neglected under specific circumstances without affecting the overall functionality. The main reason behind is that most applications already have the ability to sustain under measurement or rounding errors, and thus slight SEU-induced errors can be simply regarded as having similar tolerable ef-

fects.

M-FM's specific definition still depends on applications. For example, in a simple arithmetic circuit like Adder, we define an occurrence of error if and only if the difference between faulty and correct value exceeds a certain *threshold* value provided by user, which is shown in Equation (4.1).

$$|VALUE_{faulty} - VALUE_{golden}| > Threshold \tag{4.1}$$

In other examples like DSPs, the error can be defined based on deviations in time or frequency domain. As M-FM takes application's specific functionality along with user's requirements into consideration, its analyzing result better approximates the real situations.

4.2 Impact on Circuit Reliability

AMFT affects the circuit reliability from another perspective compared with ABFT. AMFT redefines what is error. By applying the application-dependent fault model M-FM, we're able to map and encode the application specific tolerance information into logic level to redefine error. While L-FM is an overestimation of the circuit's error rate since it does not distinguish the importance of different kind of errors, AMFT agrees well of really important errors by taking user demand into consideration.

Table 4.1 compares the difference of error rate estimation between the two fault models, L-FM and M-FM, for the DSP application as simple as 32-bit Adder. The second and third columns list the circuit's estimated error rate respectively, and the last column shows their difference. It is shown that at the threshold value 16, which is still a small number considering the 33-bit wide output, L-FM over estimates errors by 14.71%. Note that we perform the comparison based on M-FM, as it reveals the robustness of the application under real working environment. The comparison actually shows to what extent the difference can be by taking or without taking application's specific flexibility or fault-tolerant ability into consideration.

AMFT also provides us with a structural way to distinguish the error critical

Table 4.1: Comparison between L-FM and M-FM for 32-Bit Adder

Threshold	L-FM	M-FM	Diff
1	0.0156	0.0152	2.63%
2	0.0156	0.0149	4.70%
4	0.0156	0.0146	6.85%
8	0.0156	0.0140	11.43%
16	0.0156	0.0136	14.71%

level of different parts in the circuit. Specifically, LUTs in the fanin cone of lower output bits are less vulnerable to errors, since LSBs tend to be less important than other outputs in calculating the addition value. Figure 4.1 graphically verifies this approach. By ordering SRAM bits based on fanin cones from LSB to MSB outputs, we find that the criticalities of bits in fanin cones of LSBs are nearly zero. Through this structural estimation, we can avoid the time-consuming fault simulation process.

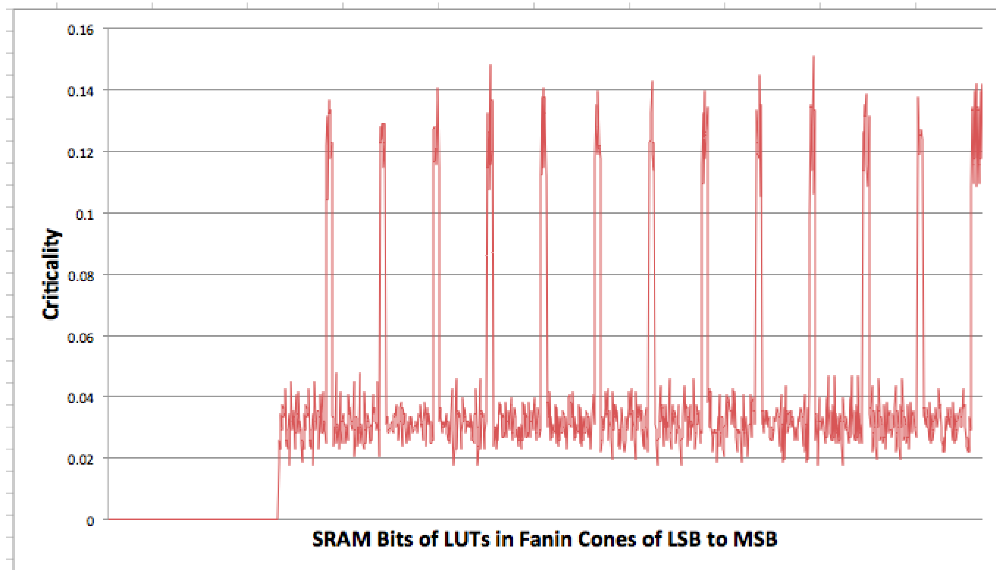


Figure 4.1: Criticality of Different SRAM Bits in 32-Bit Adder. The Bits are Ordered Based on Fanin Cones from LSB to MSB outputs.

4.3 Case study of AMFT: Discrete Convolution

With recent increase in performance and size, DSP has becoming an important area where FPGAs have found many applications [46]. Among various applications, Discrete Convolution is the single most important technique in DSP [49]. By using convolution, we can construct the output of system for any arbitrary input signal, if we know the impulse response of system.

Given an input signal $x[n]$ and impulse response function $h[n]$, the output signal $y[n]$ is simply computed as:

$$y[n] = \sum_{j=0}^{M-1} h[j]x[n-j] \tag{4.2}$$

where M is the length of $h[n]$.

Convolution illustrates a wide range of applications in which the sequence of output values at different time must be considered as a whole. More specifically, in the case of any error, we're more concerned with the similarities between faulty and correct waveforms rather than the difference in any single data point. For example, when we analyze the time domain response of the digital system, certain variations can be neglected as noise without affecting the system behavior. In many applications, we care much more about the shape instead of the amplitude [49].

In our experiment, we use correlation coefficient [35] as a metric to quantitatively measure the similarity between two waveforms X and Y:

$$\rho = \frac{cov(X,Y)}{\sigma_x \sigma_y} \tag{4.3}$$

where $cov(X,Y)$ is the covariance of X and Y, σ_x and σ_y are their standard deviations respectively. Mathematically, we have $0 \leq |\rho| \leq 1$, where 1 stands for perfect match. To reflect the significance of SEU on different configuration bits, we compute the average correlation of faulty and golden output for each bit. Then, we define the *criticality* for bit b, which measures the severeness of system error if that bit is flipped due to SEU, using:

$$Crit_b = 1 - |\rho| \tag{4.4}$$

Similar to Section 2.5, the error rate of LUT or entire circuit can be calculated as the average criticality of all bits it contains.

The detail experiment flow is shown in Figure 4.2. To map algorithm information, we propose to optimize the circuit's robustness through partial TMR technique [44]. Different from [48] and [47] which improve circuit reliability by replicating certain components at macro block level, we rank all LUTs through fault simulation and try to TMR top 10% error critical LUTs. Since a relatively small portion of the circuit is triplicated, the area and power overhead is much less than full TMR [40]. The same with former study case starting from verilog description, we transform it into BLIF format using Altera's QUIP toolkit [45]. Then, L-FM and M-FM are used separately to generate the top 10% critical LUTs for partial TMR. Finally, we compare the reliability improvement.

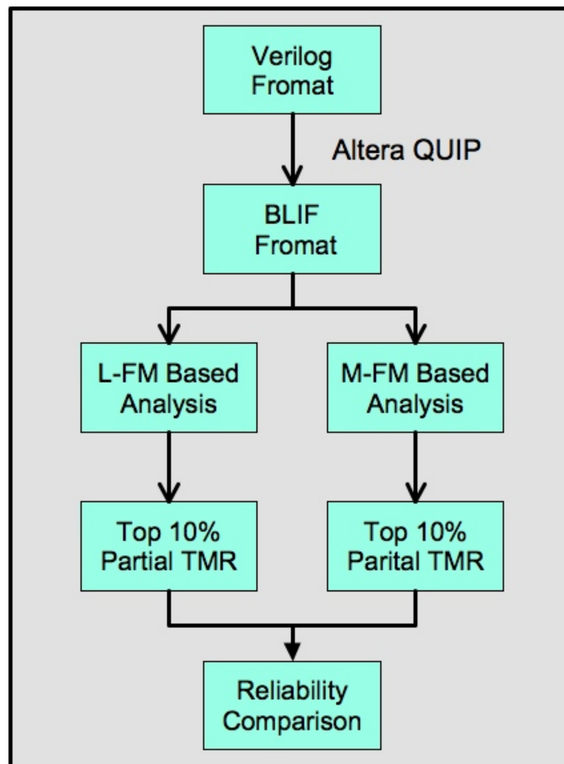


Figure 4.2: Experimental Flow

Table 4.2 compares the result of applying partial TMR to top 10% critical LUTs calculated by L-FM and M-FM respectively. While M-FM increases circuit reli-

Table 4.2: Comparison between L-FM and M-FM Based Partial TMR

Method	Error Rate	Ratio
Before TMR	0.041	1x
L-FM Based TMR	0.038	0.93x
M-FM Based TMR	0.012	0.29x

ability by 3 times, L-FM based method barely got any enhancement. Apparently, L-FM misinterpreted the critical level of configuration bits, since it considers each output independently and loses the overall picture of the system's functionality.

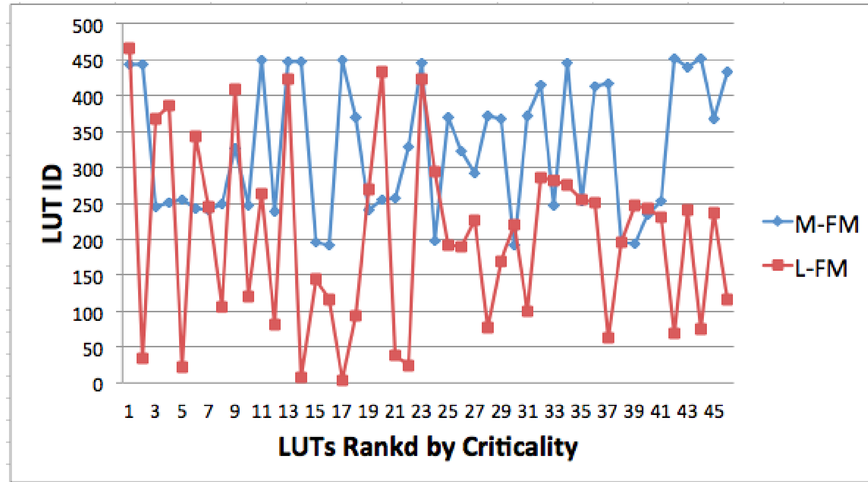


Figure 4.3: Difference in selected LUTs for TMR

To make things worse, L-FM destroys the fidelity or distribution for critical element ranks. Since most optimization techniques work in a greedy way to optimize most error critical elements first [36] [42], such degradation in fidelity makes the optimization working in wrong directions, i.e., trying to optimize less error critical LUTs first.

Figure 4.3 further illustrates the difference of L-FM and M-FM to select LUTs for TMR. The horizontal axis is the rank of LUTs sorted by the criticality from high to low, and the vertical axis represents the index of LUTs in that rank. It is observed that without taking AMFT into consideration, L-FM tends to find less critical LUTs actually, which is the reason leading to little TMR improvement.

Chapter 5

Conclusions

In this thesis, we have quantitatively studied the impact of utilizing application-level fault tolerance to improve circuit robustness. Except for considering the architecture-level and logic-level flexibilities to achieve certain reliability of the circuits, AFT should be considered with particular application. Two key DSP applications, the matrix multiplication and discrete convolution, are analyzed to quantitatively illustrate the effect of algorithm-based fault tolerance and algorithm-mapping fault tolerance respectively. Experimental results verify the large difference in circuit reliability with and without considering these high-level flexibilities. More specifically ABFT reduces the circuit's error rate by 18x for the matrix multiplication application and using our newly proposed AMFT-based fault model M-FM, 3x improvement in circuit reliability is achieved for discrete convolution, which is a key operation in DSP applications. With considering ABFT, the error rate of the output has been decreased without too much space and power overhead in TM-R. AMFT can map the property of application into the definition of criticality and further provide more accurate direction for other fault tolerance mechanisms.

5.1 Future Research Directions

In the future, we plan to further encode the application-level fault tolerance mathematically, and feed such information to improve logic synthesis. In addition, we'd like to analyze the fault-tolerant problem within certain application domains, with the hope for domain specific optimization approaches.

Bibliography

[1] P.E. Dodd and L.W. Massengill. Basic mechanisms and modeling of single-event upset in digital microelectronics. *IEEE Transactions on Nuclear Science*, 50(3):583– 602, 2003.

[2] Single Event Upset. Altera Corp. http://www.altera.com/support/devices/reliability/seu/seu-index.html

[3] F L Kastensmidt, L Carro, and R Reis. *Fault-Tolerance Techniques for SRAM-Based FPGAs*, volume 32. Springer, 2006.

[4] M. Ceschia, M. Violante, M.S. Reorda, A. Paccagnella, P. Bernardi, M. Rebaudengo, D. Bortolato, M. Bellato, P. Zambolin, and A. Candelori. Identification and classification of single-event upsets in the configuration memory of sram-based fpgas. *Nuclear Science, IEEE Transactions on*, 50(6):2088 – 2094, dec. 2003.

[5] P. Graham, M. Caffrey, J. Zimmerman, and E. Johnson. Consequences and categories of SRAM FPGA configuration SEUs. In *International Conference, Washington DC*, volume 11, page 2003, 1909.

[6] M. Ohlsson, P. Dyreklve Neutron Single Event Upsets In SRAM-based FPGAs In Xilinx Corp. AppNote, 98

[7] K. Chapman and L. Jones. SEU strategies for Virtex-5 devices. *Xilinx Corporation, XAPP864*, 2009.

[8] J. Cong and K. Minkovich. LUT-based FPGA technology mapping for reliability. In *Proceedings of the 47th Design Automation Conference*, pages 517–522. ACM, 2010.

[9] Dan Alexandrescu, Lorena Anghel, and Michael Nicolaidis. New methods for evaluating the impact of single event transients in vdsm ics. *Symposium A Quarterly Journal In Modern Foreign Literatures*, 2002.

[10] S H Crain, J E Mazur, R B Katz, R Koga, M D Looper, and K R Lorentzen. Analog and digital single-event effects experiments in space, 2001.

[11] J F Leavy, L F Hoffmann, R W Shovan, and M T Johnson. Upset due to a single particle caused propagated transient in a bulk cmos microprocessor. *Response*, 38(6):1493–1499, 1991.

[12] K Joe Hass. Probabilistic estimates of upset caused by single event transients. *Energy*, pages 6–8, 1999.

[13] K Joe Hass, Jody W Gambles, Bill Walker, and Mike Zampaglione. Mitigating single event upsets from combinational logic. *Vlsi Design*, pages 1–10, 1998.

[14] K Mohanram. Simulation of transients caused by single-event upsets in combinational logic. *IEEE International Conference on Test 2005*, pages 973–981, 2005.

[15] Allan H Johnston. Scaling and technology issues for soft error rates. *Most*, (October):1–9, 2000.

[16] M V O'Bryan, K A LaBel, R A Reed, J W Howard, R L Ladbury, J L Barth, S D Kniffin, C M Seidleck, P W Marshall, C J Marshall, and et al. *Radiation damage and single event effect results for candidate spacecraft electronics*, pages 106–122. IEEE, 2000.

[17] E Normand. Correlation of inflight neutron dosimeter and seu measurements with atmospheric neutron model, 2001.

[18] J.A. Zoutendyk, L.D. Edmonds, and L.S. Smith. Characterization of multiple-bit errors from single-ion tracks in integrated circuits. *Nuclear Science, IEEE Transactions on*, 36(6):2267 –2274, dec 1989.

[19] R A Reed, M A Carts, P W Marshall, C J A Marshall Marshall C J, O A Musseau Musseau O, P J A Mcnulty McNulty P J, D R A Roth Roth D R, S A Buchner Buchner S, J A Melinger Melinger J, and T A Corbiere Corbiere T. Heavy ion and proton-induced single event multiple upset. *IEEE Nuclear and Space Radiation Effects Conference*, 44(6):2224–2229, 1997.

[20] R. Velazco, D. Bessot, S. Duzellier, R. Ecoffet, and R. Koga. Two cmos memory cells suitable for the design of seu-tolerant vlsi circuits. *Nuclear Science, IEEE Transactions on*, 41(6):2229 –2234, dec. 1994.

[21] Xilinx. Virtex 2.5V field programmable gate arrays. *Data Sheet*, Xilinx, Sanjose, CA, April 2001.

[22] Xilinx. Virtex Architecture Guide. *Tech. Rep.*, Xilinx, Sanjose, CA, September 2000.

[23] R.J. Francis. A tutorial on logic synthesis for lookup-table based fpgas. In *Computer-Aided Design, 1992. ICCAD-92. Digest of Technical Papers., 1992 IEEE/ACM International Conference on*, pages 40 –47, nov, 1992.

[24] D.G. Marvis. *A Reconfigurable, Nonvolatile, Reprogrammable, Radiation-hardened Field Programmable Gate Array (FPGA) for Space Applications*. Mission Research Corporation, 2001.

[25] J. Canaris and S. Whitaker. Circuit techniques for the radiation environment of space. In *Custom Integrated Circuits Conference, 1995., Proceedings of the IEEE 1995*, pages 77 –80, may 1995.

[26] J.J. Wang, R.B. Katz, J.S. Sun, B.E. Cronquist, J.L. McCollum, T.M. Speers, and W.C. Plants. Sram based re-programmable fpga for space applications. *Nuclear Science, IEEE Transactions on*, 46(6):1728 –1735, dec. 1999.

[27] Gustavo Neuberger, Fernanda De Lima, Luigi Carro, and Ricardo Reis. A multiple bit upset tolerant sram memory. *ACM Transactions on Design Automation of Electronic Systems*, 8(4):577–590, 2003.

[28] Gustavo Neuberger, Fernanda De Lima, Luigi Carro, and Ricardo Reis. A multiple bit upset tolerant sram memory. *ACM Transactions on Design Automation of Electronic Systems*, 8(4):577–590, 2003.

[29] Carl Carmichael, Earl Fuller, Joe Fabula, and Fernanda De Lima. Proton testing of seu mitigation methods for the virtex fpga. *Design*, 3(x):1–7, 2001.

[30] Anthony Salazar and Los Alamos National Laboratories. Correcting single-event upsets through virtex partial configuration. *XAPP216*, 216(March):1–12, 2000.

[31] XILINX INC. Viretx Series Configuration Architecture User Guide. *XAPP 151*, USA, 2000.

[32] D. Brand. Redundancy and don't cares in logic synthesis. *Computers, IEEE Transactions on*, 100(10):947–952, 1983.

[33] R.K. Brayton and F. Somenzi. Boolean relations and the incomplete specification of logic networks. In *VLSI89*, 1989.

[34] M.L. Case, V.N. Kravets, A. Mishchenko, and R.K. Brayton. Merging nodes under sequential observability. In *Proceedings of the 45th annual Design Automation Conference*, pages 540–545. ACM, 2008.

[35] Correlation and Covariance of a Random Signal. http://cnx.org/content/m10673/latest/

[36] Z. Feng, Y. Hu, L. He, and R. Majumdar. IPR: In-Place Reconfiguration for FPGA fault tolerance. In *Computer-Aided Design-Digest of Technical Papers, 2009. ICCAD 2009. IEEE/ACM International Conference on*, pages 105–108. IEEE, 2009.

[37] M. Gardner. Mathematical games: The fantastic combinations of John Conway's new solitaire game 'Life'. *Scientific American*, 223(4):120–123, 1970.

[38] Y. Hu, Z. Feng, L. He, and R. Majumdar. Robust FPGA resynthesis based on fault-tolerant Boolean matching. In *Computer-Aided Design, 2008. ICCAD 2008. IEEE/ACM International Conference on*, pages 706–713. IEEE, 2008.

[39] K.H. Huang and J.A. Abraham. Algorithm-based fault tolerance for matrix operations. *Computers, IEEE Transactions on*, 100(6):518–528, 2006.

[40] F.L. Kastensmidt, L. Sterpone, L. Carro, and M.S. Reorda. On the optimal design of triple modular redundancy logic for SRAM-based FPGAs. In *Proceedings of the conference on Design, Automation and Test in Europe-Volume 2*, pages 1290–1295. IEEE Computer Society, 2005.

[41] Berkeley Logic Synthesis and Verification Group. ABC: A System for Sequential Synthesis and Verification, Release 70930. http://www.eecs.berkeley.edu/ alanmi/abc/

[42] J.Y. Lee, Z. Feng, and L. He. In-place decomposition for robustness in FPGA. In *Computer-Aided Design (ICCAD), 2010 IEEE/ACM International Conference on*, pages 143–148. IEEE.

[43] CA Lisboa, L. Carro, C. Argyrides, and DK Pradhan. Algorithm level fault tolerance: a technique to cope with long duration transient faults in matrix multiplication algorithms. In *VLSI Test Symposium, 2008. VTS 2008. 26th IEEE*, pages 363–370. IEEE, 2008.

[44] B. Pratt, M. Caffrey, P. Graham, K. Morgan, and M. Wirthlin. Improving FPGA design robustness with partial TMR. In *Reliability Physics Symposium Proceedings, 2006. 44th Annual., IEEE International*, pages 226–232. IEEE, 2006.

[45] Quartus II University Interface Program.

[46] J. Serrano. Digital signal processing using field programmable gate arrays.

[47] B. Shim and N.R. Shanbhag. Energy-efficient soft error-tolerant digital signal processing. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 14(4):336–348, 2006.

[48] B. Shim, S.R. Sridhara, and N.R. Shanbhag. Reliable low-power digital signal processing via reduced precision redundancy. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 12(5):497–510, 2004.

[49] S.W. Smith. *Digital signal processing: a practical guide for engineers and scientists*. Newnes, 2003.

[50] S. Yamashita, H. Sawada, and A. Nagoya. A new method to express functional permissibilities for LUT based FPGAs and its applications. In *Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design*, pages 254–261. IEEE Computer Society, 1997.