UNDERSTANDING AND MITIGATING THE EFFECTS OF SOFT ERRORS IN LOGIC

by

Eric L. Hill

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Electrical Engineering)

at the

UNIVERSITY OF WISCONSIN–MADISON

2009

Committee's Page. This page is not to be hand-written except for the signatures

A dissertation entitled

Understanding and Mitigating the Effects of Soft Errors in Logic

submitted to the Graduate School of the
University of Wisconsin-Madison
in partial fulfillment of the requirements for the
degree of Doctor of Philosophy

by

Eric L. Hill

Date of Final Oral Examination: **December 18th, 2008**

Month & Year Degree to be awarded: **December** **May** **2009** **August**

****

Approval Signatures of Dissertation Committee

Kewal K. Saluja

Michael Scheinfein

Signature, Dean of Graduate School

Committee's Page. This page is not to be hand-written except for the signatures

Abstract

This thesis focuses on gaining a deeper understanding of how radiation induced transient faults, or soft errors, affect the operation of, and more importantly the high-level design decisions related to logic dominated components of a computer system. The work completed in this thesis is motivated by several trends. First, continued technology scaling has caused soft error rates to rise to a level where reliability is a concern in design spaces outside of the server domain. Second, the majority of deployed solutions within current chips are intended to protect storage structures, meaning that a growing fraction of the transistors on die vulnerable to faults belong to logic blocks. Third, studies done on an architectural level utilize performance tools, which are at a level of abstraction where the implementation details of logic blocks is unavailable. These tools generally model soft errors in both storage and combinational logic elements in the same manner. The combination of these trends indicate the need for additional investigation with regards to the effects of soft errors in logic, and more specifically how these effects impact architectural design decisions.

The work completed in this thesis represents a successful attempt at gaining

a greater understanding of these effects. The experiments conducted uncover several surprising and counterintuitive insights relating to this subject, including the appropriate manner in which comparisons relating to reliability should be made, the level of detail in which faults should be modeled, and the manner which transient faults manifest themselves. These insights are valuable in that they serve to refine the intuition of architects with regards to how various design decisions affect the reliability of logic. These insights also can be used to drive the assumptions made by tools at higher levels of abstraction when modeling transient faults. Additionally, this thesis explores how the insights gained can be leveraged in order to determine the best strategy to protect a particular logic component.

Acknowledgments

I would first like to thank my parents, Frank and Darlene Hill, for the unconditional support they provided throughout my life and specifically during the completion of my PhD. Their constant optimism and encouragement definitely helped me get through the rough times in my graduate career. I would also like to thank my brother, Tyrone, for his support during this process. By virtue of being in graduate school at the same time as me, he was always able to relate to what I was going through, and really helped me put things in the proper perspective.

The mentoring and support provided by my adviser, Professor Mikko Lipasti, played an integral role in my personal and professional development during my time in Madison. His general approach to evaluating and solving problems has definitely shaped me as an engineer, and his unwavering optimism and encouragement drove me to be persistent and complete my degree. Additionally, I would like to thank the other members of the PHARM research group for the assistance they provided in helping me complete this work. I would specifically like to thank Ilhyun Kim for providing me with his scalar z80 decoder design, which was used as a starting point for many of the studies performed in this thesis.

I would also like to thank Kelly Burton and Professor Doug Henderson, the coordinators of the Graduate Engineering Research Scholars (GERS) program at UW-Madison. They provided constant support for me throughout my time in Madison, and were always there when I needed guidance or advice.

I would not be anywhere near as prepared for the professional world without the internship opportunities I was given during my graduate career. I would like to thank Lisa Wu for the mentoring she provided during my two graduate internships at Intel. From her I specifically learned the level of dedication and commitment required to succeed in a professional environment.

I also benefited greatly from the interactions I had with other students and faculty members during my time at UW. I would like to thank my thesis committee for the feedback they provided on my research, and specifically Kewal Saluja for the additional insights he provided when my interests shifted towards fault-tolerance. The graduate students I interacted with during my time here not only influenced me from a technical perspective, but also helped me achieve some degree of work-life balance. During my time in graduate school I was fortunate enough to make many lasting friendships that I hope will endure even when I leave Madison.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

As computing systems become increasingly ubiquitous, architects strive to create robust systems capable of operation in a wide range of environments. In addition to meeting performance and power requirements, engineers now have to spend a significant amount of time ensuring their designs also meet reliability goals. The combination of continued technology scaling and increased on-chip transistor densities have made vulnerability to radiation-induced transient faults (soft errors) a significant design concern [35]. Soft errors were initially a problem in high density memory cells, first being observed in DRAMs and then later on in SRAM-based caches[83]. This initial discovery led to a significant number of proposed solutions designed to prevent, detect, and/or correct faults occurring in storage cells.

One consequence of this pervasive protection of storage structures is that an increasingly large fraction of the vulnerable transistors on die belong to combina-

tional logic blocks. In addition to this, failure rates due to transient faults on logic nodes are predicted to increase by several orders of magnitude due to technology scaling [61]. For these reasons, engineers will need to devote additional design effort to protecting logic in order to meet reliability goals in future systems.

Recent research on soft errors has largely been divided into two domains. In the architecture domain, most proposals either offer some form of global thread-level redundancy [8][34][66] or monitor storage structures which hold sensitive micro-architectural state [10][73][35]. In the context of these schemes, logic dominated units are either not protected or are essentially replicated (either spatially or temporally) as part of a larger redundancy scheme. The majority of work in this domain relies on performance simulation tools for evaluation, with latch-accurate RTL models used in a minority of cases[70]. Modeling the effects of soft errors at this level is difficult, as structural and timing characteristics of logic blocks are not available.

In contrast, there exist a significant number of proposals in the implementation domain with the purpose of mitigating the effects of transient faults in combinational logic. A myriad of techniques [81][14][38][32][13] have been proposed in this domain. Techniques presented in this context typically use gate and/or transistor-level models for evaluation, allowing for the effects of circuit structure and timing constraints to be considered. Unfortunately, modeling industrial sized circuits at this level of detail is often computationally intractable.

This dissertation has two overarching objectives. The first objective is to gain a more detailed understanding of transient fault propagation characteristics

in combinational logic blocks typically found in high performance microprocessors. The primary motivation for this is to allow architects to accurately reason about the effects of soft errors in logic earlier in the design cycle, essentially bridging the gap between the two previously described domains of soft error research. The second objective of this thesis is to leverage the aforementioned fault propagation characteristics in order to explore cost-effective means of protecting logic blocks from transient faults. These objectives were successfully accomplished through the completion of several steps. First, simple, intuitive fault models were developed to facilitate reasoning about how transients propagate. Next, several studies were performed to understand how low-level structural and timing characteristics could potentially affect high level design decisions. Finally, fault propagation characteristics discovered from the previous steps of this thesis were used to develop cost effective soft error protection techniques.

1.1 Thesis Contributions

In this dissertation, several contributions are made. With respect to gaining a deeper understanding of the soft error problem within logic, the work completed in this thesis provides several new insights related to answering the following questions:

- How should architects conceptually think about this problem? The results presented in this dissertation show that contrary to conventional intuition, the vulnerability of a given logic block is largely independent of pipeline

depth. In addition to this, combinational logic gates within more deeply pipelined circuits are actually less vulnerable to transient faults.

- What effects are the most important to model? Several existing proposals on methodologies to model soft errors in logic only focus on modeling the logical propagation of errors, modeling timing effects analytically. The results presented in this work show that this simplification can lead to misleading results with respect to how the impact of transients faults varies with clock frequency.

- How are the artifacts of transient faults structured? Many reliability studies performed at the architecture or application level of abstraction model the final result of a transient fault in logic in the same manner as SRAM, as a single bit flip. The analysis conducted in this dissertation shows that when transients faults are studied using a gate-level infrastructure, this single bit flip assumption is not always valid. Specifically, it is shown that in many cases a single transient fault can result in multiple state bits being corrupted, and that state bits are not corrupted with equal probability.

The insights uncovered in this thesis related to these questions contrast with prevailing intuition and are particularly beneficial to architects and others working at higher levels of abstraction. Additionally, this thesis explores how the answers to these questions can be practically applied in terms of protecting individual logic blocks.

1.2 Thesis Organization

The remainder of this thesis is divided into six chapters. Chapters 2 and 3 provide basic background related to soft errors along with a description of the tools developed to conduct this study, respectively. Chapter 4 is devoted to exploring the how scaling clock frequencies and pipeline depths affect the soft error vulnerability of a given logic block. Chapter 5 primarily focuses on systematically outlining which protection schemes are best for a given logic block, and also presents characterization results for a parallel instruction decoder regarding the structure of the artifacts produced by transient faults. Chapter 6 presents a novel framework for transient fault analysis, which uses the fault propagation characteristics of a circuit in order to provide error tolerance in a cost effective manner. Finally, Chapter 7 outlines the conclusions reached in this thesis and outlines various avenues of future work.

Chapter 2

Background

2.1 Introduction

The purpose of this chapter is to provide additional background related to what physically occurs when a transient fault is induced, as well as historically how the soft error problem evolved to be a concern for general purpose architects. The remainder of this chapter is divided into four sections. The first section describes the mechanics of how transient faults affect memory cells, in addition to providing some historical background on the problem. The second section provides more details on how transient faults manifest themselves in logic. The final two sections describe metrics used to express soft error rates, and provide a qualitative argument about why logic soft error rates are expected to rise to a level equivalent to unprotected SRAM.

2.2 Soft Errors

Radiation induced transient faults, or soft errors, typically originate from two sources. Soft errors can be caused by alpha particles present in packaging materials, or by neutron particles from cosmic radiation. While alpha particle induced errors were previously identified as a serious problem in high density memories, neutron particles are the primary source of errors in current generation systems [83][35]. A soft error occurs when a radiation particle strikes the bulk of a transistor, generating some amount of charge. The exact amount of charge generated by particle is primarily dependent on its energy. If a sufficient amount of charge is absorbed by the source and/or drain region of the affected transistor, a single event effect is induced, meaning that the value stored at that particular circuit node is flipped. If the affected transistor is part of a memory cell, this corruption is known as a single event upset (SEU). If the affected transistor is part of a combinational logic gate, the fault is known as a single event transient (SET)[61]. The minimum amount of generated charge necessary to induce a single event effect is generally denoted as the critical charge or Q_{crit}[15]. This Q_{crit} value is primarily dependent on the sizing of the transistors within the gate or memory cell of interest, as this directly affects the capacitance stored at each circuit node. A lower Q_{crit} value indicates that a component is less reliable, as it implies that a larger fraction of striking particles will be able to generate enough charge in order to induce a single event effect.

While a soft error related failure in current generation process technology is

likely to originate from a particle strike either on a logic gate or a memory cell, historically memories have been significantly more susceptible to faults. Moore's Law, an empirical observation which describes the rapid increase in transistor integration density over time[33], has transformed the phenomena of soft errors from an issue that was originally only of concern in the high availability server or avionics application spaces [8][75][68] to something that architects of general purpose systems now worry about.

In previous technology generations, soft error rates were low enough such that the effects of transient faults were only noticeable in high density memory components. The early appearance of error correcting codes (ECC) and parity logic first in DRAMs and later in large SRAM caches is evidence of this initial concern. The continued shrinking of transistor dimensions, which is generally viewed as a benefit of Moore's Law, also has the negative effect of reducing the minimum amount of charge required to induce a single event effect, effectively increasing the soft error rate with every technology generation.

Eventually soft error rates rose to a level that the effects of transient faults started to become observable not only in high density memory components, but also within storage structures commonly found within conventional microprocessor pipelines. At this point, controlling soft error rates became a topic of increased interest in the architectural research community. This renewed interest inspired numerous proposals on ways to model and understand the effects of soft errors at an architectural level [37][70][35]. Because these techniques arose out of the architecture community, the majority of these proposals model soft errors

using binary instrumentation tools, performance simulators, and in some cases behavioral RTL models.

In parallel with the previously described observation of errors within architectural storage structures, soft error rates due to particle strikes on combinational logic gates also increased dramatically. In the past naturally occurring masking phenomena prevented a large fraction of SETs from ever resulting in errors. The rate at which SETs are masked is decreasing rapidly due to technology scaling. Essentially, logic soft error rates are not only rising due to lower Q_{crit} due to shrinking geometries, but also because of diminished rates of masking. The previously mentioned architectural studies use tools at levels of abstraction such that an understanding (on an architectural level) of the effects of soft errors in logic is precluded.

2.3 Soft Errors in Logic

Recall from the previous section that a particle strike affecting a combinational logic gate manifests itself as a transient pulse at the gate output. In order for an SET to have the same result as an SEU, which would be to invert the value in a storage bit, the transient pulse needs to logically propagate through the circuit and ultimately alter the value captured by a downstream sequential element. In previous technology generations, logic soft error rates were kept low due to natural masking phenomena that prevented this described scenario from occurring. Each of these masking phenomena will now be discussed in detail, along with how

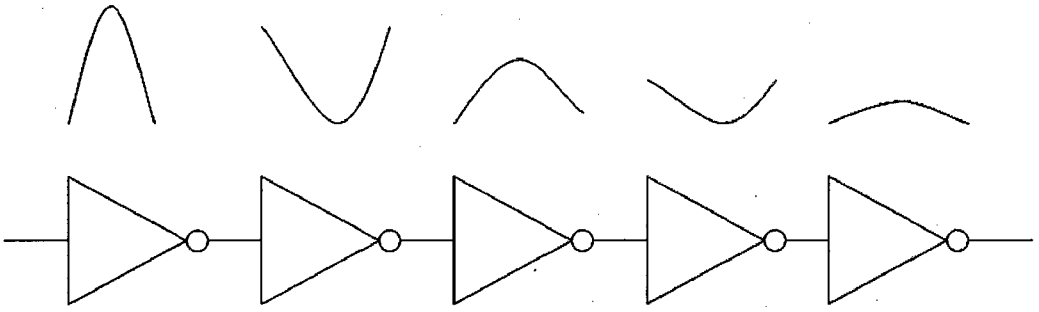


Figure 2.1: Example of Electrical Masking.

scaling is affecting its significance.

Electrical Masking

As a transient pulse propagates through a combinational logic chain, some attenuation occurs (in terms of the height and width of the waveform) as the transient passes from the input to the output of each gate. If the height of the pulse is degraded to the point where the transient does not represent a logically altered value, or the width is shortened to the point where the transient value does not persist long enough to meet the setup + hold time required alter the value captured by a sequential element, the transient is said to be electrically masked[61]. The amount of electrical masking that occurs at each gate is a function of the propagation delay of that gate as well as the width of the transient pulse[39]. An example of a transient pulse being attenuated as it propagates through a logic chain is shown in Figure 2.1. The amplitude and duration of the transient pulse decreases as it propagates through the successive gates in the logic chain.

The general trend of decreasing clock cycle times reduces the effect of elec-

trical masking because there are fewer levels of logic per pipeline stage, meaning that a transient pulse propagates through fewer downstream gates before arriving at a sequential element. In addition to this, decreasing feature sizes across successive technology generations impact the effect of electrical masking in two ways. First, decreasing feature sizes imply that transistors will have lower Q_{crit} values, meaning that transient waveforms will increase in size (in terms of height and width). Also, because propagation delays decrease with smaller feature sizes, the amount of waveform attenuation that occurs at each individual gate will be diminished.

Logical Masking

When a particle strikes a device in a combinational logic chain, the resulting transient pulse is only in danger of corrupting a downstream sequential element if the chain is on a sensitized path. Whether or not a path is sensitized is a function of the inputs applied to the logic block. Consider the logic shown in Figure 2.2. The transient present at the output of gate A is eliminated because the controlling value at the other input of gate B prevents it from propagating any further.

Transient faults blocked in this manner are said to be logically masked [61]. The amount of logical masking that occurs within a logic block is a property of the function computed and is independent of all technology parameters.

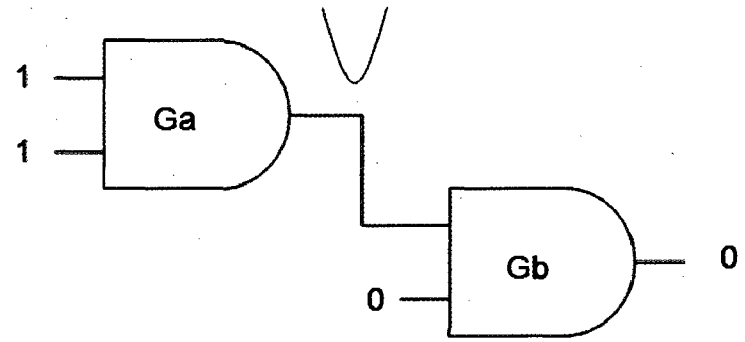


Figure 2.2: Example of Logical Masking.

Timing Window Masking

Even if a radiation particle hits a gate on an appropriately sensitized path, and generates a transient waveform with sufficient height and width to corrupt the value captured by a downstream sequential element, the transient must arrive at the input of the sequential element during the period where the sequential captures a new value. This period of time is defined by the setup and hold time of the sequential in question, and is generally referred to as the latching window [61][76][44]. Pulses that reach the inputs of sequentials outside of this time period are said to be timing window masked. Figure 2.3 illustrates different timing window masking scenarios that might occur.

For this figure, it is assumed that the sequential element being corrupted is a positive-edge triggered flip-flop, meaning that the latching window is centered around the rising edge of the clock. In Figure 2.3, the waveform at the top of the illustration represents the clock signal, and the dotted lines represent the latching window. As shown in this figure, a transient only alters what is captured by a

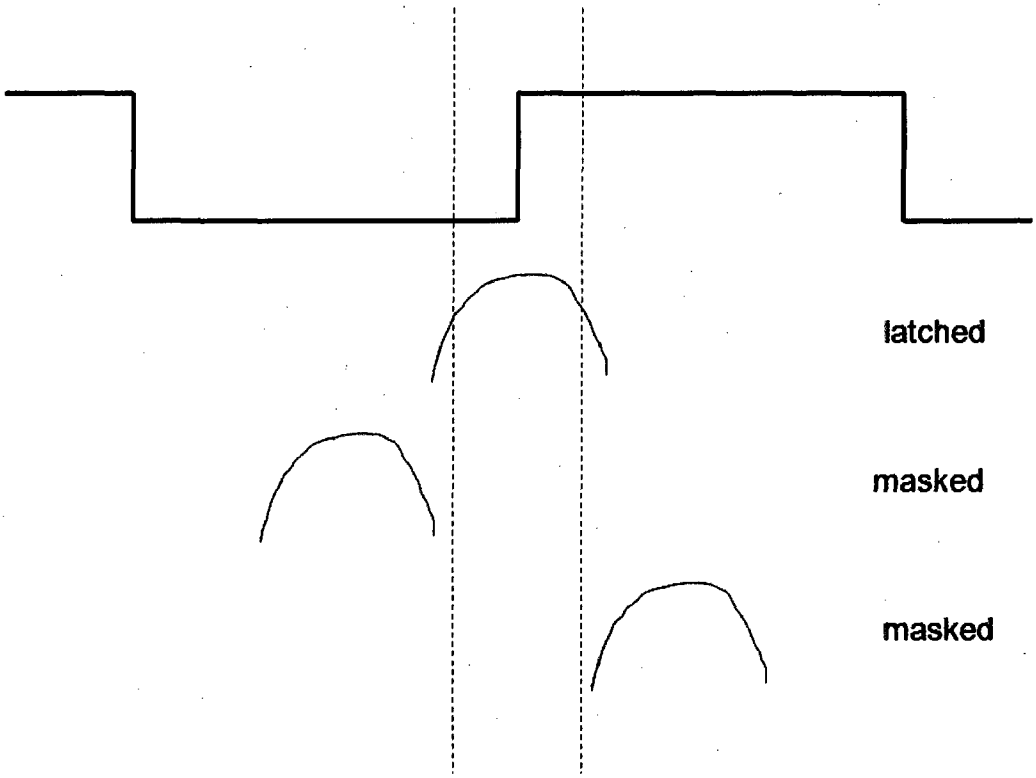


Figure 2.3: Example of Timing Window Masking.

flip-flop when the transient value is present at the data input of the flip-flop during the latching window. Continued technology scaling will decrease the amount of timing-window masking occurs. Shrinking Q_{crit} values mean that propagating transients will have greater widths, and shrinking gate delays mean that both clock periods and latching windows will be shorter. The combination of these two effects increase the probability that a transient arriving at a flip-flop input will do so during the latching window, meaning that timing-window masking will occur less often.

2.4 Reliability Metrics

Soft error rates are generally expressed using the metric of Failures in Time (FIT), which is defined as the number of failures per 10^9 hours [35]. In some situations, the metric of mean time to failure (MTTF) which is the inverse of FIT, is also used to express failure rates. In general, the FIT rate can be calculated in the manner shown in Equation 2.1.

$$FIT = (RawStrikeRate) * (Derating) \tag{2.1}$$

The FIT rate of a system is typically calculated by multiplying the Raw Strike Rate, and a derating factor. Because neutron particles are the dominant source of soft errors, the Raw Strike Rate can be approximated by the expression shown in Equation 2.2.

$$SER \propto F * A * exp(-Qcrit/Qs) \tag{2.2}$$

Equation 2.2 represents the soft error rate as a function of altitude dependent neutron flux (F), vulnerable drain area (A), and the ratio of (-Q_{crit}/Q_s), where Q_s is the charge collection efficiency, and is property the the transistor affected by the particle strike [19]. The derating factor is defined as the probability that a particle strikes manifests itself as an visible output error. What actually factors in determining the derating factor is largely dependent on the definition of a visible error. In the context of a logic block, a visible error might be defined as the case

where a fault results in incorrect values computed at the outputs of that block. In this case the derating value would be determined by the degree that electrical, logical, and timing window masking occur within the block. In the context of an entire system, a visible error might be defined as the case where a fault results in a divergence in committed architectural state. In this case, the degree of architectural masking within a system would also play a role in determining the derating factor.

2.5 Relationship Between Logic and SRAM Soft Error Rates

The combination of shrinking feature sizes and diminishing amounts of timing window and electrical masking occurring imply that logic soft error rates will increase by several orders of magnitude over the next several technology generations. Both of these trends have the ultimate effect of reducing the average Q_{crit} for a circuit, meaning the radiation particles with less energy will be able to induce SETs. A relatively recent study on the logic soft error scaling has predicted that as a consequence of both of the the aforementioned trends, logic soft error rates will be comparable to unprotected SRAM error rates by the 50nm technology generation[61]. A plot of estimated Q_{crit} values from this study is plotted in Figure 2.4. In this graph, the estimated Q_{crit} values for SRAM cells, combinational logic gates are plotted, along with the charge collection efficiency Q_s. The lines with diamond, square, and triangle markers represent the values

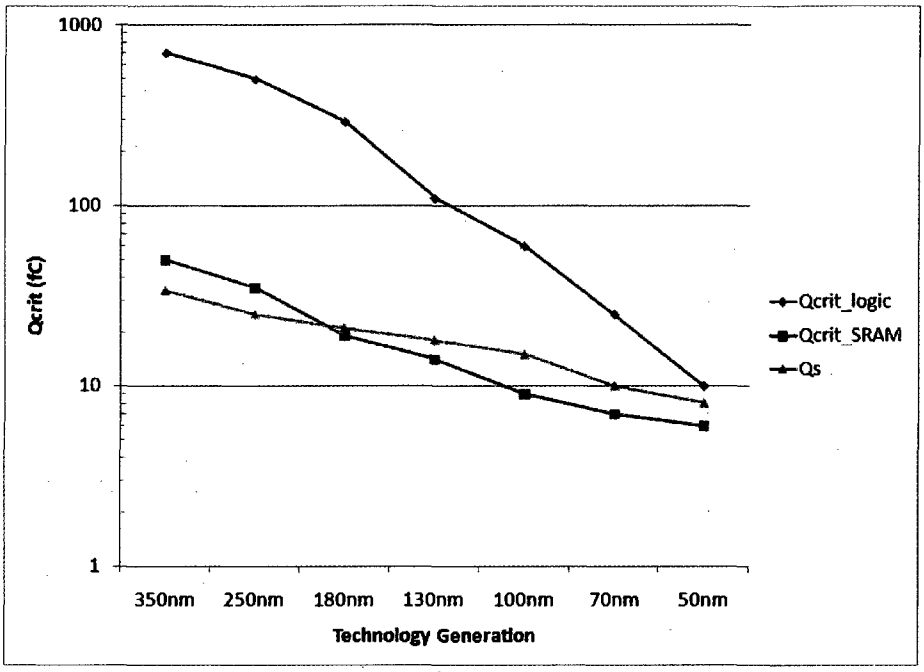


Figure 2.4: Scaling of Q_{crit} with Process Technology. Adapted from [61].

of Q_{crit} for logic, Q_{crit} for SRAM, and Q_s respectively. This plot can be used to qualitatively explain why technology scaling will cause logic error rates to significantly increase.

Recall from Equation 2.2 the soft error rates have an exponential relationship with the ratio (Q_{crit}/Q_s). In Figure 2.4, the Q_{crit}_SRAM is plotted in the context of an individual transistor within an SRAM cell, while Q_{crit}_logic is shown in the context of a transistor within a logic chain, meaning that the plot of this quantity also includes the effects of electrical and timing window masking. It is clear from the plot shown in Figure 2.4 that Q_{crit}_logic is decreasing significantly faster than Q_{crit}_SRAM across technology generations. The underlying reason

behind the trend shown in the plot is that Q_{crit}_SRAM is only decreasing as a result of diminishing feature size, while Q_{crit}_logic is decreasing because of the combined effects of smaller features sizes and less timing and electrical masking. Essentially the quantities of Q_{crit}_SRAM/Q_s and Q_{crit}_logic/Q_s are converging to the same value.

2.6 Summary

In this chapter, background information relating to soft errors was provided. In particular, the historical background regarding the context in which the effects of transient faults were first observed was described, as well as the underlying physical processes that occur in order for a particle strike to induce an actual fault. Finally, this chapter concluded by providing a qualitative rationalization as to why logic soft error rates are projected to increase faster than SRAM error rates.

Chapter 3

Methodology

3.1 Introduction

In this chapter, the infrastructure developed to evaluate the effects of soft errors in logic is described. In order to develop this infrastructure, several trade offs had to be made with respect to the level of detail used to model faults. While a certain level of detail is required in order to draw useful conclusions from experiments performed, too much detail makes the study of any logic block large enough to be of interest computationally intractable. The strategy taken in this dissertation is to study the effects of soft errors in logic through statistical fault injection. The actual modeling done is split into two parts, a circuit level modeling component with the purpose of obtaining realistic characteristics of transient pulses, and logic level modeling to accurately capture transient propagation characteristics within a given logic block. The rest of this chapter is divided into four sections. The

first section provides an overview of prior proposals for modeling transient faults. Following this, the next two sections discuss the circuit and logic-level modeling performed for this work, respectively. Because the tool chain developed for this study relies on statistical fault injection, the last section of this chapter is devoted to establishing proper experiment lengths in order to ensure the statistical validity of later results collected with this framework.

3.2 Related Work

While statistical fault injection is intuitively the most straightforward approach to obtaining reliability estimates, in many cases it can be an extremely time consuming methodology. Many logic blocks have a large state space of operating conditions under which faults may naturally occur. In the context of transient fault modeling, this space at the very least has several dimensions: the range of all possible inputs, all vulnerable circuit nodes, all points in time during the clock cycle, and all possible particle energies. For this reason, there have been numerous proposals aimed at reducing the time required for fault simulation by removing one or more dimensions of this state space.

In this vein there have been several techniques proposed which reduce the input dimension through either symbolically modeling circuit nodes and propagating faults with binary decision diagrams (BDDs)[9][30], or bit-parallel simulation[23]. Parallel simulation has also been applied with respect to the particle-energy dimension, with several existing proposals advocating the simultaneous simulation

of transients with a range of waveform characteristics[46][79].

Perhaps the most commonly utilized strategy for reducing the computational requirements is to eliminate the timing dimension. Proposals using this simplification typically model the effects of logical masking in detail, while modeling timing-window masking and in some cases electrical masking analytically [76][44][61] [5][23]. One advantage of this particular strategy is that many tools in the ATPG domain are designed to measure the degree to which the values at circuit nodes are observable at outputs, and can be trivially extended to model the effects of logical masking.

For the purposes of this work, the framework constructed does not explicitly remove any of the previously described state space dimensions. The rationalization for this design decision was that not removing any dimensions would yield the most accurate results, and that the availability of machines for parallel simulation would allow for experiments to be conducted in a reasonable amount of time.

3.3 Circuit-Level Modeling

Recalling the background discussion in Chapter 2, particle strikes on combinational logic gates manifest themselves as transient pulses (or glitches) occurring at the output of the affected gate. The most important characteristics of these transient waveforms are the amplitude and duration of these pulses, which are primarily dependent on the size of the affected transistor as well as the energy

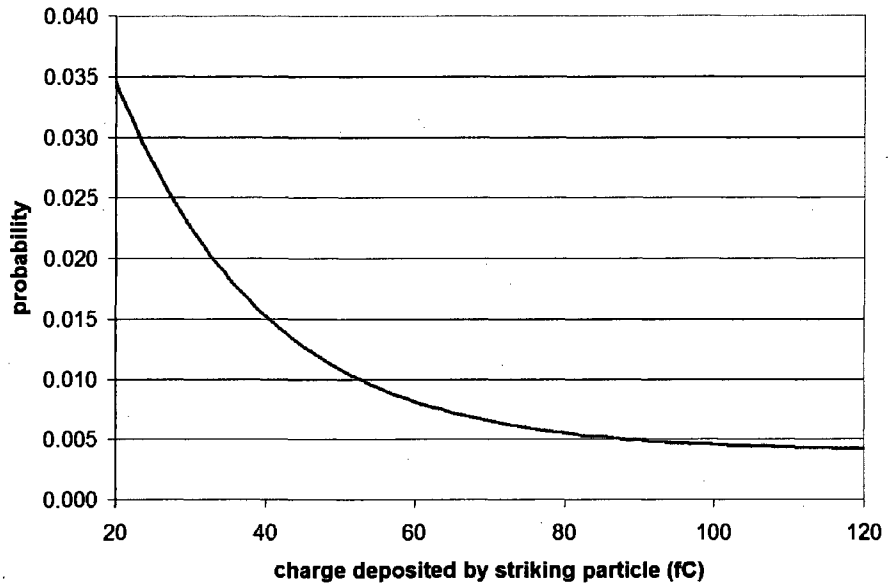


Figure 3.1: Charge Deposition PDF. From [13].

of the striking particle. The purpose of the circuit-level modeling component of this study was to establish realistic ranges for these quantities.

Several prior studies on soft errors in logic have simulated particle strikes by first modeling a combinational CMOS gate at a transistor level, and then injecting pulses of current, simulating the charge collection process, into drain nodes within that gate and observing the transient waveforms that occur at the the gate outputs [13][61]. This methodology was also used for the modeling done in this dissertation work. These experiments were conducted using HSPICE using the 65 nm predictive technology model [2]. The shape of the injected pulse was modeled as a time-dependent exponential function, as described by [61]. The

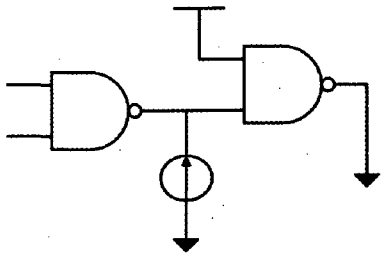


Figure 3.2: NAND Structure Used for SET Waveform Characterization.

function used is shown in Equation 3.1.

$$I(t) = Q/T * \sqrt{t/T} * \exp(-t/T) \tag{3.1}$$

This equation describes a time dependent current pulse as a function of the charge generated by a striking particle (Q), and a technology dependent time constant for charge collection (T). The time constant used corresponds to the value given in [61] that most closely corresponds to a 65 nm process. A range of values for Q was obtained from a previous published probability density function for charge deposition shown in Figure 3.1. Figure 3.2 illustrates the experimental setup used for particle strike simulation. This test bench mimics the methodology used in [13].

In terms of modeling, the most important transient waveform characteristics that need to be captured are waveform height and duration. Instead of tracking and storing both of these characteristics, the framework developed for this study defined and measures SET duration, which is defined as the period of time the transient wave form is above or below the value of Vdd/2 (a supply voltage of

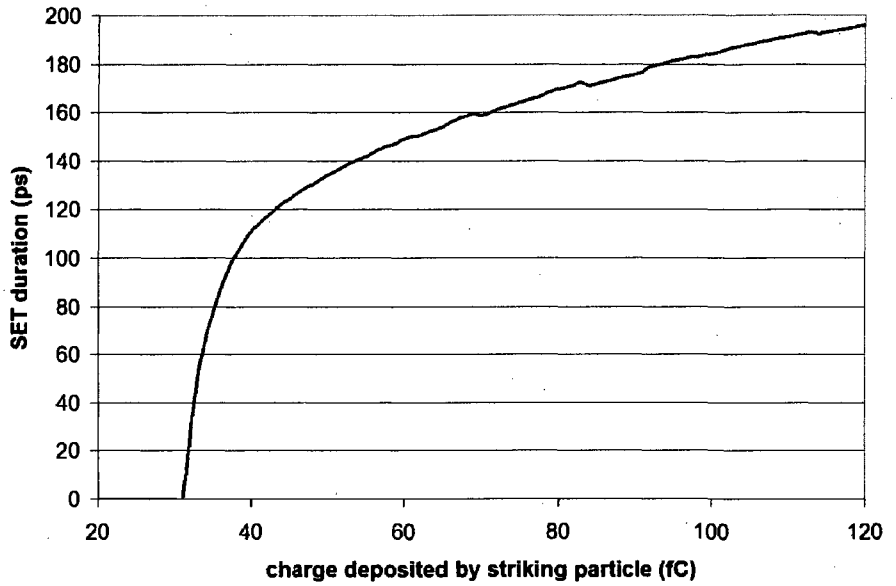


Figure 3.3: SET Duration Mapping Function.

0.9 V was used), corresponding to a logically different value.

The ultimate goal of the described circuit modeling experiments was to sweep across the range of possible amounts of charge being generated, and determine the corresponding SET durations. The results of this exercise are shown in Figure 3.3, with the range of charges used represented on the x-axis, and the corresponding SET durations plotted on the y-axis. These SET duration values were then mapped to the x-axis of the probability density function shown in Figure 3.1, creating a new probability density function for SET duration. This newly created function was then discretized and used as an input to the gate-level fault injection component of the developed framework.

3.4 Gate-Level Modeling

The second component of the framework developed for this study is a gate-level simulator used for statistical fault injection. This simulator was created by adding timing support to an existing tool originally intended for automatic test program generation (ATPG). Additionally, the 5-valued logic alphabet, originally proposed by Roth [55] and typically used for ATPG, was extended to include 7 values in total giving the simulator the additional ability to model faults within flip-flops.

Each fault injected can be represented by a 4-tuple, with members representing the gate or flip-flop affected, the point in the clock period during which the particle strike occurs, the input vector applied to the logic block, and the duration of the generated SET. When faults are injected, the time that the fault occurs as well as the input vector applied to the logic block are chosen randomly. For the input vectors, additional functionality is added to the simulator which allows certain inputs to be fixed to predetermined values. This is especially useful during the evaluation of logic blocks with control inputs, eliminating input combinations that would never occur in practice and outside of the intended functionality of the logic block. SET durations are chosen according to the discrete probability density function, whose construction was previously described.

Realistically choosing gates and flip-flops to inject faults into presented an additional challenge. Because the developed simulation infrastructure was derived from ATPG tools, the logic blocks evaluated must be represented in the UW net

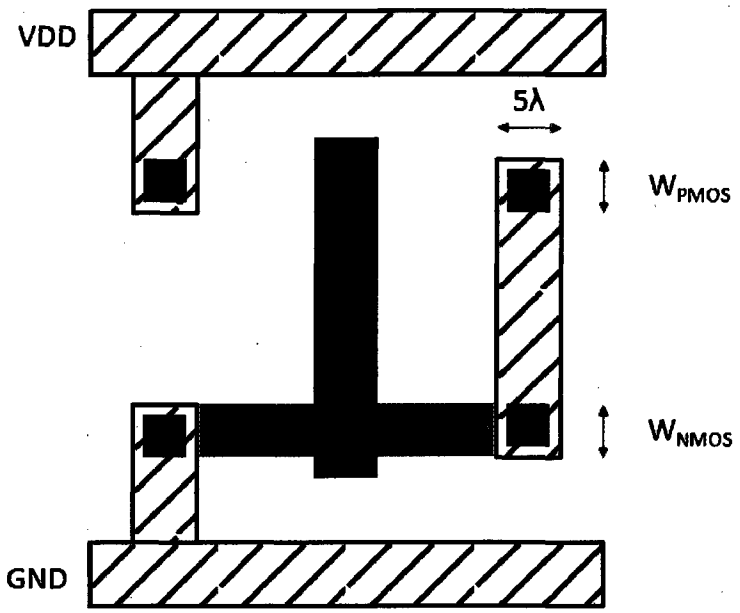


Figure 3.4: Example Inverter Layout for Area Estimation.

list format, which is derived from the more commonly used ISCAS format [17]. This format represents gates as one of several elementary types (NAND, NOR, XOR, BUF, DFF, etc...) with no notion of area. A first order area model was developed by creating simple layout level diagram of each gate, and estimating the vulnerable drain area.

An example stick diagram of a CMOS inverter is shown in Figure 3.4. By constructing diagrams like this for each gate type, drain area can be easily calculated. In Figure 3.4 both the length and width of the diffusion regions are specified in terms of a constant λ. VLSI design rules are typically specified in terms of λ, which usually represents half of the minimum feature size of a given process. By specifying rules in terms of this variable, migrating design rules

of one process generation to the next is greatly simplified [74]. The length of the diffusion regions modeled was assumed to be 5λ, which was suggested as a reasonable value by [74].

Gate Type)	Delay (ps)	Drain Area (nm^2)
NAND	21	84500
NOR	21	105625
AND	39	147875
OR	39	169000
INV	17	63375
DFF	n/a	126750

Table 3.1: Gate Characteristics.

The delay of each type of gate was calculated through the use of logical effort. The logical effort of a gate is defined as the ratio of its input capacitance to the input capacitance of an equivalently sized inverter[67]. This ratio is useful because it allows the delay of a more complex gate to be easily approximated from the delay of an inverter. Table 3.1 shows the different types of elementary gates modeled, along with the assumed propagation delays and vulnerable drain area.

The diagram in Figure 3.5 shows all of the possible outcomes that can occur when a particle strike affects a combinational logic gate. Looking at this figure, there are several different classes of outcomes possible for each fault. Outcomes A and E represent cases where the transient is logically masked, either before corrupting a flip-flop (A), or after (E). Outcomes B and D represent cases where a SET reached the input of a flip-flop, but not during the rising clock edge, meaning

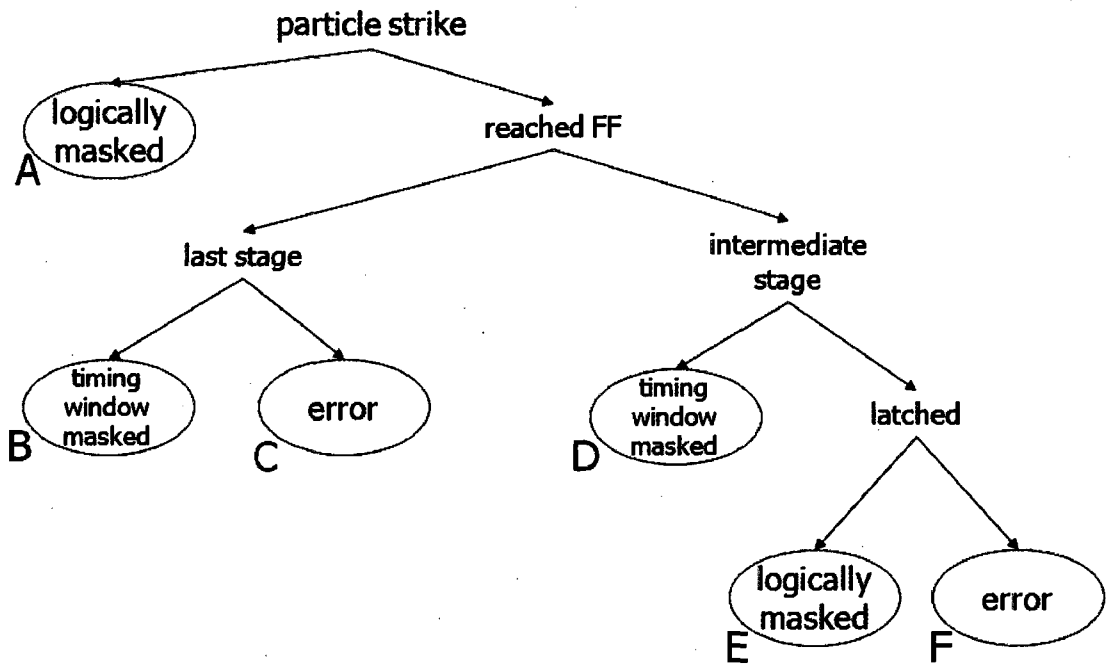


Figure 3.5: SET Fault Outcome Tree.

the fault was timing window masked. Only outcomes C and F actually represent cases where an injected fault results in incorrect bits at the primary outputs of a circuit. From this diagram it should be clear that while the developed framework does account for logical and timing window masking effects, electrical masking is not modeled. For this work the choice was made to ignore the effects of electrical masking for two reasons. Because it is well understood that the degree a combinational gate can attenuate a propagating transient pulse is dependent on both the delay of the gate as well as the width of the transient [39][61][79][44],

the amount of observed electrical masking will be essentially constant across all experiments (since no gate sizing is being performed). Any effects of electrical masking will be canceled out when comparisons are performed. In addition to this, prior proposals exist which model the effects of electrical masking by constructing piecewise equations which relate the waveform characteristics of a transient waveform at the output of the gate to the input waveform as well as the delay of the gate [39]. Generally, if a transient has a pulse width that is significantly larger than the delay of the gate it is passing through, the fault will propagate unattenuated. During the previously described circuit level characterization experiments, the observed transient widths were significantly larger than the gate delays used. This relationship should hold as process technology continues to scale.

In addition to modeling faults due to particle strikes on transistors within combinational logic gates, the infrastructure developed for this dissertation is also capable of modeling particle strikes within storage cells within flip-flops. For this case, it is assumed that a particle strike will only corrupt the SRAM cell used for storage within either the master or slave latch. It is assumed for this work that all flip-flops are constructed by placing 2 level sensitive latches back to back. Furthermore, in the developed gate-level simulator, each flip-flop is only modeled as a single SRAM cell. This design choice was made because of a previous observation made by Siefert, which is that an individual latch is only vulnerable to particle strikes in opaque state [59]. Assuming a single phase clocking scheme with a 50 % duty cycle, implying that only one SRAM cell is

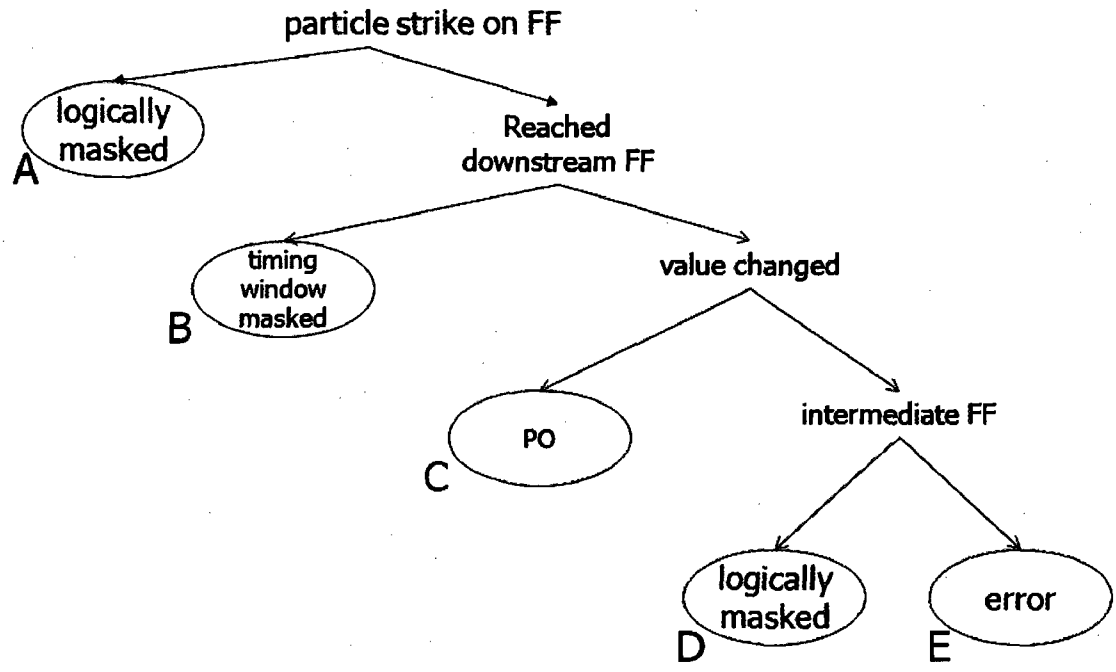


Figure 3.6: SEU Fault Outcome Tree.

vulnerable at any given time.

A diagram of the possible outcomes that can occur when a latch is affected by a particle strike is shown in Figure 3.6. The outcomes shown in this diagram were conceptually derived by considering the abstraction of a 1-bit wide arbitrary datapath, beginning and terminated with single flip-flops, denoted as launching and receiving, respectively. For each outcome, it is assumed that one latch in the launching flip-flop is affected by a particle strike. It should be noted that in cases where latches are affected by particle strikes, the erroneous values that propagate

through the rest of the circuit are not transient pulses, but rather permanent values which will persist until the next clock cycle. Using this single bit datapath abstraction, outcome A in Figure 3.6 represents the case where the launching flip-flop is affected by a particle strike, but no erroneous values propagate to the receiving flip-flop because of logical masking. Outcome B represents the case where the launching flip-flop is affected by a particle strike that is capable of logically propagating to the receiving flip-flop, but this is prevented from happening, because the particle strike occurred too late in the clock cycle. This scenario is most likely to occur on long paths within a circuit and was explicitly characterized in [59], where each flip-flop in the circuit was assigned a window of vulnerability (WOV), bounding the portion of a clock cycle where a SEU would have enough time to propagate to a downstream receiving flip-flop.

Outcomes C, D, and E represent the cases where the launching flip-flop is affected by a particle strike within its designated WOV, and the erroneous values ended up propagating to at least one receiving flip-flop downstream. Outcome C represents the case where the receiving flip-flop is also a primary output, meaning that an error has occurred. Outcome C also includes a special subset of cases where a primary output flip-flop is directly affected by a particle strike. Outcomes D and E represent cases where the receiving flip-flop is not a primary output, and erroneous values are either logically masked later (D) or eventually affect primary outputs (E).

3.5 Benchmark Creation

In order to be evaluated using the tool chain developed for this work, Verilog modules were synthesized to LSI10k standard library cells using Synopsys Design Compiler Version Y-2006.06-SP1. These synthesized modules were then translated to the required UW net list format via perl scripts.

3.6 Statistical Significance of Fault Injection

Because the tool chain developed for this dissertation work relies on statistical fault injection for data collection, a lower bound on the number of faults to inject in order to generate results with some degree of statistical validity needs to be established. This lower bound of injected faults is established through the construction of confidence intervals. Ultimately the result produced by the tools whose development is described in this chapter is a derating value, which is the number of errors observed, where an error is defined as a case where an injected faults results in incorrect values at circuit outputs, divided by the total number injected. The true derating value of a circuit can only be obtained with absolute certainty if faults were injected under all input, location, timing, and particle energy conditions, a task which is computationally intractable. The statistical fault injection tools developed for this work inject faults only under a subset of all possible conditions, essentially producing a sample measurement which is an approximation of the true quantity. Confidence intervals provide a measurement of the representativeness of these measurements by establishing an

interval centered around the sample value as well a probability that the true value lies within that interval.

Confidence intervals are typically defined in terms of the size of the interval (usually expressed as the percent difference between the ends of the interval and the sample value), and a confidence level (denoted as α), where the probability that the true value lies within the defined interval is (1 - α) [20].

For the purposes of our study, the bounds of our interval are defined by the expression shown in Equation 3.2.

$$\bar{x} \pm t_{\alpha/2}(n-1) * (s/\sqrt{n}) \quad (3.2)$$

This equation shows the construction of an interval dependent on the mean of the sample measurement ($\bar{x}$), the measured standard deviation (s), a distribution and confidence level dependent constant ($t_{\alpha/2}(n - 1)$), and the number of samples taken (n). The statistical methodology used for this study were based primarily on the discussion of confidence intervals in [20]. The most important decision made was the use of the t-distribution rather than the normal distribution, which was made due to the small number of samples used, as well as the lack of prior knowledge regarding the distribution of the values being estimated.

For the purposes of this study, the most important variable in Equation 3.2 is n, which directly represents the number of faults that need to be injected in order to achieve a desired confidence interval. Rearranging the terms for the upper bounds of the confidence interval (as shown in Figure 3.2), an expression for n

can be obtained. This expression is shown in Equation 3.3.

$$n = \left(\frac{s * t_{\alpha/2}(n - 1)}{\text{upper bound} - \bar{x}}\right)^2 \tag{3.3}$$

Determining a Minimal Sample Size

In order to determine the minimal sample size required to have statistically valid results, several characterization experiments were performed. For this study, the OpenSPARC floating point multiplier was used as a benchmark [29]. This circuit was chosen because it is the largest circuit studied in this dissertation work, containing over 35,000 gates and 2,000 flip-flops when synthesized to LSI10k library cells. For the first part of this experiment, 10 separate fault injection experiments were performed, with 10,000 faults injected in each case with a different random seed. For each simulation the measured derating was recorded. From these initial simulations the values of the sample mean and standard deviation (denoted by $\bar{x}$ and s, respectively) as well as the upper bound of the desired confidence interval can be calculated. With all of these values determined, the number of samples required can be obtained by applying the formula shown in Equation 3.3.

The results of this experiment are summarized in Table 3.2. The rows in this table represent the size of the interval in terms of the percent deviation from the sample mean (10% means that the lower and upper bounds would be $0.95*\bar{x}$ and $1.05*\bar{x}$, respectively), while the columns represent the $(1 - \alpha)$, or the probability that the true derating value falls within the defined interval.

	90%	95%	99%
10%	0.24	0.36	0.75
5%	0.95	1.45	2.99
1%	23.78	36.21	74.75

Table 3.2: Estimation of Required Sample Size.

The summarized results of this experiment indicate that using a sample size of 10 (corresponding to 100,000 total faults injected) is sufficiently large enough to have a 5% confidence interval with a confidence level of 99%. This essentially means that there is a 99% probability that the true derating value is within plus or minus 5% of the mean sample derating value measured using the developed tool chain. The results of this study justify the experiment lengths used for evaluation in the rest of this dissertation, where at least 30,000 faults are injected in all cases.

3.7 Summary

In this chapter, the tools and methodology used to conduct the experiments described later on in this dissertation are described. The developed tool chain combines gate-level statistical fault injection with low level circuit simulation with the goal of enabling the effects of transient faults to be modeled in a great level of detail, while allowing simulations to complete in a reasonable amount of time.

Chapter 4

Impact of Pipeline Depth

4.1 Introduction

In this chapter, the relationship between clock frequency and reliability is explored. Increasing clock frequencies are commonly cited as one of the reasons that soft errors in logic are becoming an important design concern [61][56]. The relationship between the frequency a circuit is clocked at and its soft error rate is intuitive, as shortening the clock cycle time of a circuit decreases the probability that a SET is timing window masked. The experiments conducted in this chapter show that this intuition is in fact flawed, and that the vulnerability of a logic block to soft errors is largely independent of the degree to which it is pipelined. These experiments also show that combinational gates within more aggressively pipelined circuits are more resilient to the effects of transient faults.

This study also uncovers two key observations which not only explain these

surprising results, but also serve to refine conventional intuition regarding the proper manner in which to make fair comparisons of reliability, and the appropriate level of modeling detail required to obtain realistic results. The first key result produced by this study is that the direct comparison of soft error rates is not always the appropriate manner in which to evaluate the reliability of different logic blocks. In this chapter, the scenarios where direct rate comparisons are inappropriate are outlined. In addition to this, the results presented in this chapter also underscore the importance of modeling the effects of timing window masking explicitly. The use of analytical models for timing window masking [61][9][30], obscure second order effects which can significantly impact the error rates of combinational logic.

The remainder of this chapter is divided into four sections. In the first section, the conventional intuition regarding how logic soft error rates scale with clock frequency is defined formally and validated experimentally. The next two sections identify the flaws in this intuition, and propose methodological refinements in order to address these flaws, respectively. The final section of this chapter discusses the implications of these presented results.

4.2 Conventional Intuition

Combinational Logic Soft Error Rates

As was discussed in Chapter 2, a bit flip only occurs as a result of a particle strike on a combinational logic gate when the generated SET logically propagates to

the input of a flip-flop and changes the value captured. In order for this to happen the SET has to arrive at the flip-flop data input during the rising edge of the clock. In the cases where a SET arrives at the input of a flip-flop, but not when the rising edge occurs, the SET is said to be timing-window masked. The expression shown in Equation 4.1 was proposed by [61] to analytically determine the probability that the arrival of a SET at the input of a flip-flop would coincide with the rising edge of the clock.

$$T_{derating} = \frac{(d - w)}{C} \quad (4.1)$$

Equation 4.1 expresses this probability as the function of 3 quantities: the SET duration d, which in this work is the amount of time the amplitude of the pulse is above or below Vdd/2, the latching window w, which is the sum of the setup and hold times for the flip-flop, and the clock period C. From this equation it is simple to infer the first-order effect of increasing the pipeline depth of a unit on the combinational logic soft error rate. At deeper pipeline depths, C decreases, implying an increase in $T_{derating}$. Simply put, the value of $T_{derating}$ (and thus the overall error rate) should be inversely proportional to the clock period.

Latch Soft Error Rates

In addition to being vulnerable to particle strikes on combinational logic gates, errors in computation can also occur when bits are flipped as a result of direct strikes on storage cells within flip-flops. It is assumed in this work that all flip-

flops considered are constructed of back to back level sensitive latches. Seifert et al. characterized the vulnerability of latches to particle strikes, finding that latches are only vulnerable to bit flips in opaque mode [59]. A latch in transparent mode is not vulnerable because it is being driven by fan-in logic. If it is assumed that the waveform used to clock the circuits has a 50% duty cycle, this implies that each latch is only vulnerable to a particle strike 50% of the time. When the pipeline depth of a functional unit is doubled, the first order intuition is that the latch count (and thus the latch area) should also double, implying a proportional increase in the latch soft error rate.

First Order Analysis

Following from the intuition established in the previous section, both latch and logic soft error rates should increase with pipeline depth. Several experiments were conducted with the purpose of validating the above stated intuition.

Experimental Setup

For this particular study, floating point addition and multiplication units based on the designs from the UltraSPARC T1 were chosen as benchmarks [29]. These units were chosen because nearly every general purpose microprocessor has floating point hardware. In addition to this, the decision of whether or not to fully pipeline a unit (which affects the frequency at which the unit is clocked), often comes up in the design of floating point hardware. Generating several comparison points for evaluation required the creation of multiple versions of each unit,

pipelined to varying degrees. In order to create these benchmarks, all flip-flops within the behavioral Verilog representation of these units were removed, creating purely combinational versions of each logic block. These combinational logic blocks were then synthesized to LSI 10k standard cells using Synopsys Design Compiler, and re-timed using the automatic pipelining functionality within the synthesis tool chain. This process yielded 2 stage, 4 stage, and 8 stage pipelined versions of each original circuit. The attributes of each benchmark circuit are shown in Table 4.1. The clock periods shown in this table were obtained by taking the measured critical path delay for each 8 stage design, and doubling that value successively as the number of pipeline stages is halved. While it is unlikely that the actual critical path delay would double when the number of stages is halved, the premise of this study was to consider a system where in the nominal case a functional unit is fully pipelined (into 8 stages in this case) and to explore how making the unit not fully pipelined affected reliability. The last column of Table 4.1 reports the area percentage in the context of drain regions vulnerable to particles strikes. Using the evaluation framework and methodology described in Chapter 3, statistical fault injection was performed on each benchmark, with 100,000 particle strikes being simulated in each case.

In order to properly compare the effects of pipeline depth on soft error rates, the error rates of combinational logic and latches are evaluated separately. The reason this is done is because the soft error rates are dependent on both area and derating factor. For the combinational logic case, it is sufficient to only look at derating as the overall area of gates stays roughly the same across all

Benchmark	Clock Period	flip-flops	% FF area
fpadd_comb	3040	104	1.1%
fpadd_2stg	1520	419	4.0%
fpadd_4stg	760	1123	9.9%
fpadd_8stg	380	2463	18.5%
fpmul_comb	3120	83	0.3%
fpmul_2stg	1560	389	1.5%
fpmul_4stg	780	2269	8.1%
fpmul_8stg	390	3598	12.1%

Table 4.1: Description of Benchmarks Used for Pipeline Depth Study.

pipeline configurations. In the latch case, both the area and derating factors change significantly with pipeline depth, meaning both must be considered to compare error rates.

Combinational Logic Soft Error Rates

The results of statistical fault injection into the floating point adder and multiplier circuits are shown in Figures 4.1 and 4.2, respectively. In each figure, the x-axis represents the different pipeline configurations studied for each circuit, while the stacked bars on the y-axis represent the fraction of injected faults that resulted in a particular fault outcome. A description of each fault outcome is shown in Table 4.2. From both figures, it is clear that a significant amount of injected faults never manifest themselves as errors due to either logical or timing window masking.

For the adder and multiplier, 64 and 53 percent of injected faults fall into the LM outcome in the combinational case, respectively, which is the scenario where the SET is logically masked before reaching a flip-flop input. As expected,

Outcome	Description
LSERR	SET corrupts primary output flip-flop
LSTWM	SET reaches primary output flip-flop, but is logically masked
ISLER	SET corrupts intermediate flip-flop, error propagates to primary output
ISLLM	SET corrupts intermediate flip-flop, but error is logically masked
ISTWM	SET reaches intermediate flip-flop, but is timing window masked
LM	SET is logically masked before reaching flip-flop

Table 4.2: Possible Outcomes for Combinational Logic Transient Fault Injection.

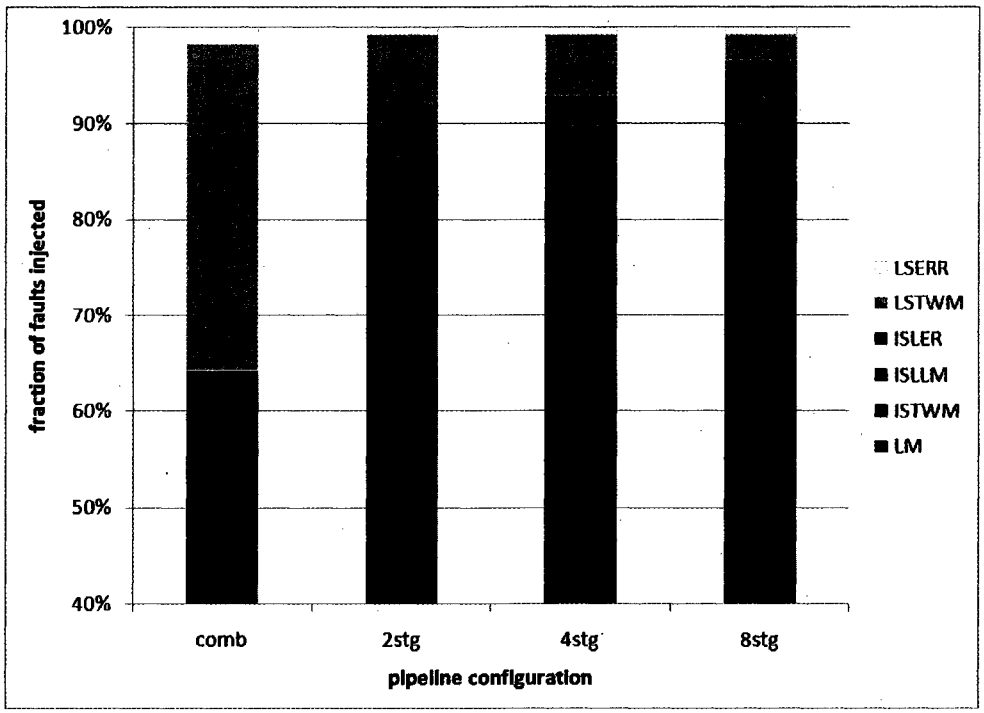


Figure 4.1: FP Adder Combinational Logic Fault Injection Breakdown.

the number of faults resulting in the LM outcome decreases as both circuits are pipelined into a larger number of stages. This decrease can be attributed to the fact that as pipeline depth increases a SET needs to propagate through fewer levels of logic before reaching a flip-flop, meaning that it is less likely to be logically

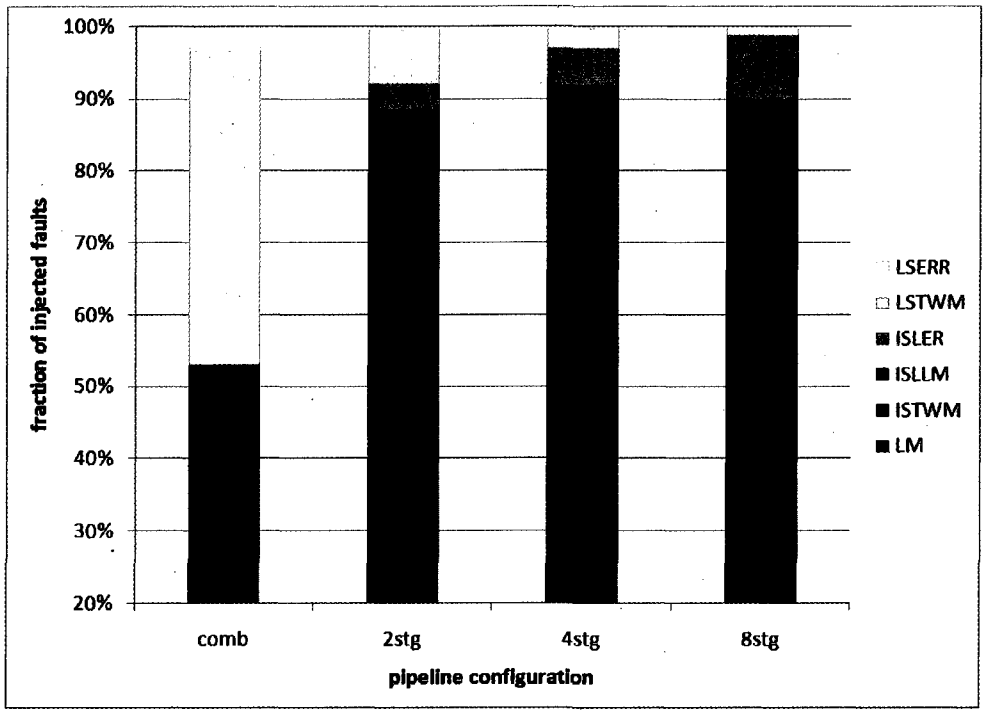


Figure 4.2: FP Multiplier Combinational Logic Fault Injection Breakdown.

masked before this happens. Correspondingly, there is an increase in ISLLM and ISLER cases, where a SET ends up corrupting at least one intermediate flip-flop, across the pipelined configurations.

The logic derating, defined as probability a particle strike on a combinational logic gate will cause the unit to compute erroneous results, is plotted in Figure 4.3. This quantity is calculated by dividing the number of faults falling into the ISLER and LSERR cases by the subset of faults injected into combinational logic gates. The derating factors presented in these plots represent the effects of both logical and timing-window masking.

As was stated previously, logical masking is an effect that should be solely

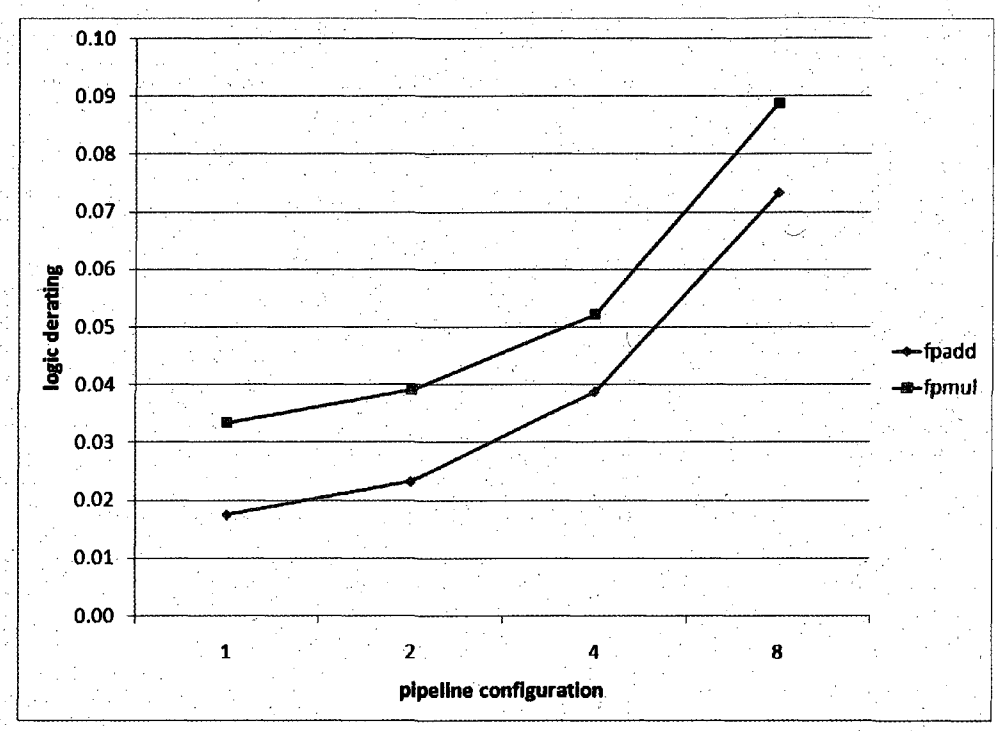


Figure 4.3: Measured Overall Logic Derating of Floating Point Units.

dependent on the function computed by a particular circuit, and therefore invariant across all pipelined configurations. The logical derating, defined for our study as the probability that an injected fault will not be logically masked, can be measured experimentally as shown in Equation 4.2.

$$\text{logical derating} = \frac{\text{LSTWM} + \text{LSERR} + \text{ISLER} + \text{ISTWM} * \frac{\text{ISLER}}{\text{ISLER} + \text{ISLLM}}}{\text{faults injected}} \tag{4.2}$$

In addition to accounting for cases where SETs logically propagate to a primary output (ISLER, LSERR, LSTWM), this equation also accounts for cases

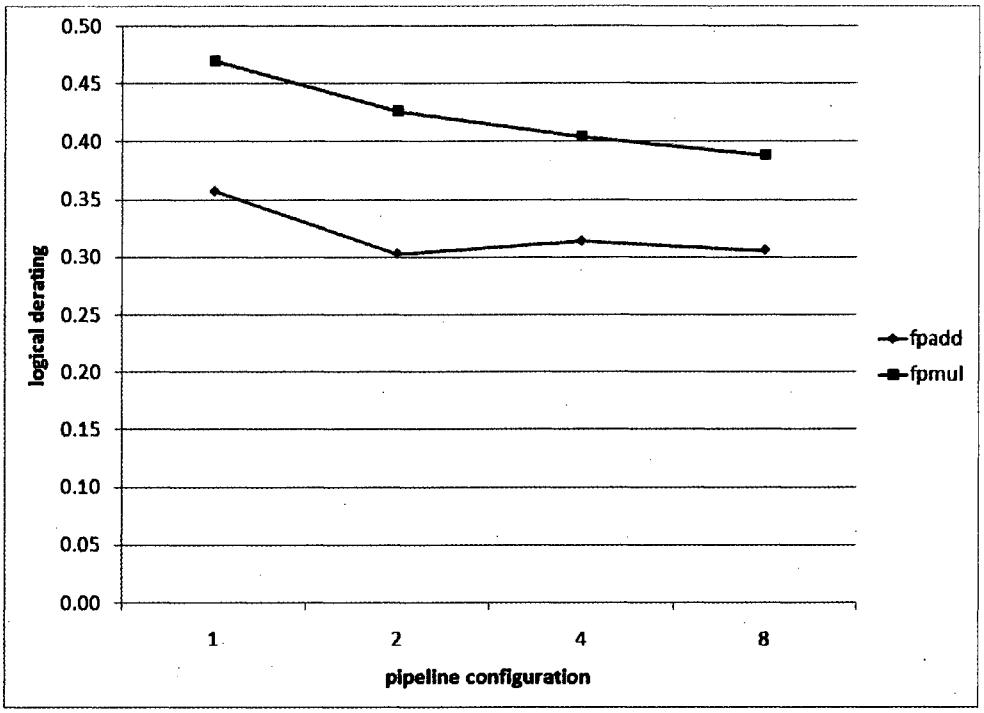


Figure 4.4: Measured Logical Derating of Floating Point Units.

where faults would have logically propagated to outputs but end up being timing window masked. The measured logical derating for both floating point circuits are shown in Figure 4.4.

Looking across the different pipeline depths, the measured logical derating is relatively stable, remaining at roughly 30 and 40% for the adder and multiplier, respectively.

$$\text{timing derating} = \frac{\text{LSERR} + \text{ISLLM} + \text{ISLER}}{\text{LSTWM} + \text{LSERR} + \text{ISTWM} + \text{ISLLM} + \text{ISLER}} \quad (4.3)$$

A significant amount of timing window masking is also occurring in both circuits. The timing derating, defined here as the probability that a SET eventually results in a bit flip, can be measured experimentally as shown in Equation 4.3. The equation shown is simply the ratio of bit flips that occur to the number of outcomes where a SET propagates to the input of a flip-flop. To separate timing window masking from logical masking, cases where a bit flip occurs and is later logically masked (ISLLM) are included in this calculation. The measured timing derating is plotted for both functional units in Figure 4.5. As expected, the probability of at least one bit flip occurring as the results of a SET increases with pipeline depth. It should also be noted that the increase in timing derating observed in Figure 4.5 is not varying linearly with clock period, as predicted by the analytical expression shown in Equation 4.1. The second order effect responsible for this nonlinear variation will be discussed in Section 4.4.

Collectively, the plots shown in Figures 4.3, 4.4, and 4.5, confirm the previously developed intuition regarding how the combinational logic soft error rate should vary with pipeline depth. Figure 4.3 shows that the error rate does indeed increase, while the combination of Figures 4.4 and 4.5 show that the increase in error rate is due to a decrease in the amount of timing window masking going on in the more aggressively pipelined versions of each circuit.

Latch Soft Error Rates

In addition to looking at the effects of particle strikes in combinational logic, another experiment was performed to understand how latch soft error rates scaled

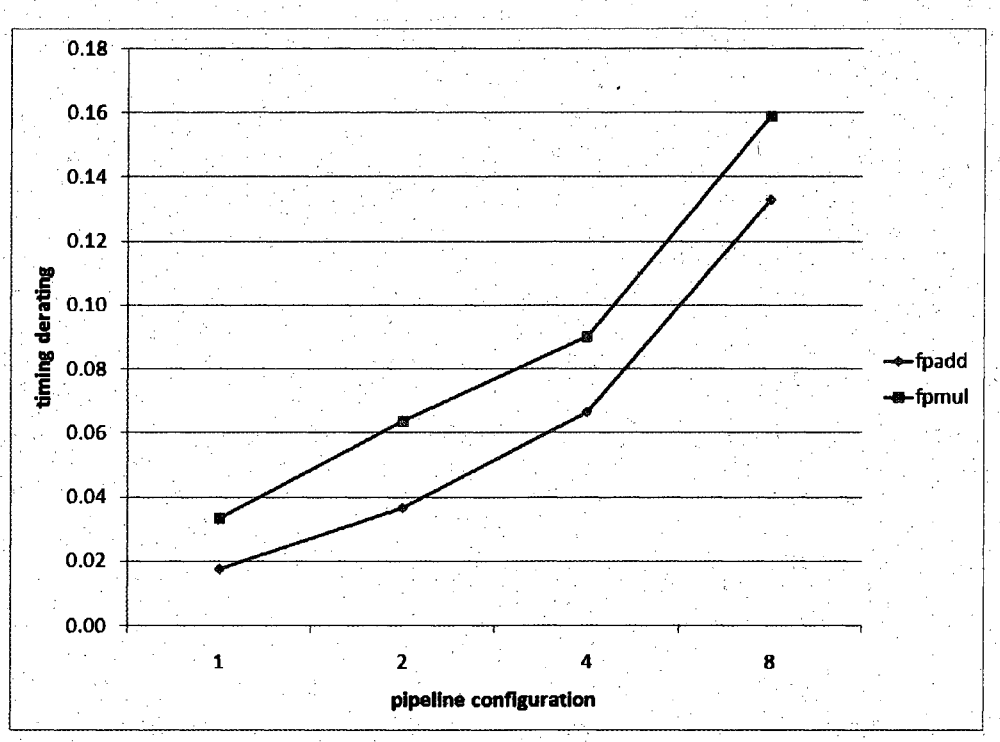


Figure 4.5: Measured Timing Derating of Floating Point Units.

with pipeline depth. In this experiment, 100,000 faults were again injected in each case, but every fault was injected into latch storage. Because the area vulnerable to particle strikes increases with the number of pipeline stages, this analysis considers the product of area and derating in order to accurately compare error rates. This is in contrast to the previous analysis shown for combinational logic, which only considered derating.

Figures 4.6 and 4.7 show the outcome breakdown of fault injection on the adder and multiplier, respectively. The x-axis in each figure represents the different pipeline configurations studied, while the stacked bars on the y-axis represent the fraction of injected faults resulting in a particular outcome. The FFPO and

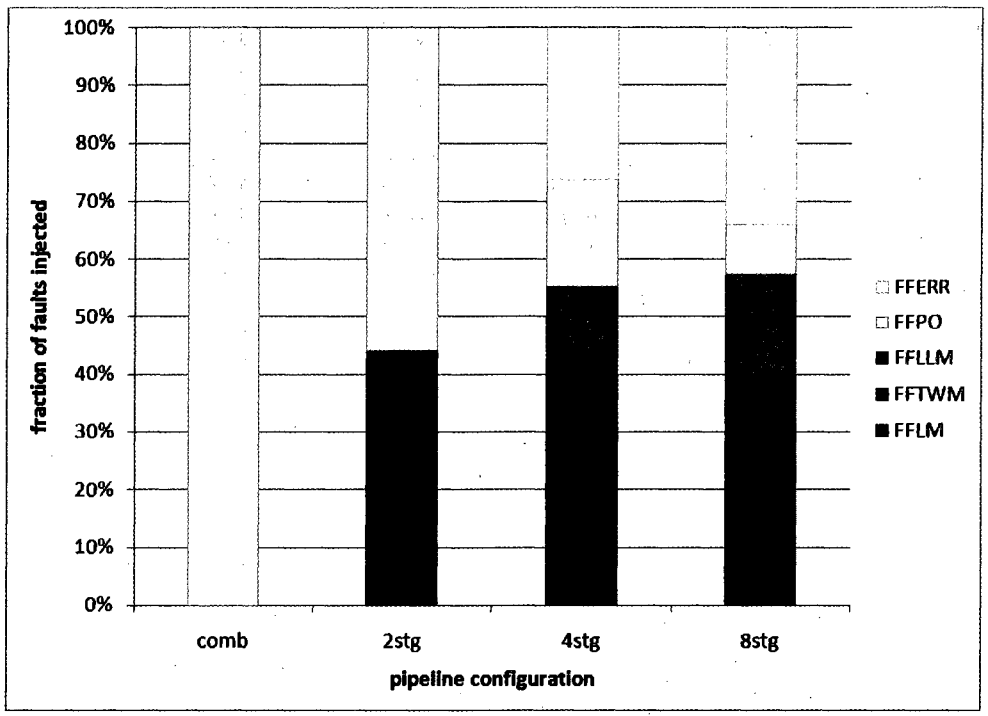


Figure 4.6: FP Adder Latch Fault Injection Breakdown.

FFERR outcomes represent errors, or cases where an injected fault ultimately results in the wrong value computed at the primary output of the circuit. For the combinational cases 100% of the faults injected result in the FFPO outcome, as the only flip-flops present in these circuits are at the primary outputs. One distinct difference between the breakdowns presented in Figures 4.6 and 4.7 and the charts previously shown for combinational logic fault injection is that there is significantly less timing window masking occurring. Recalling the discussion from the fault model presented in Chapter 3, the FFTWM outcome only occurs when a fault is injected to a flip-flop sufficiently late in the clock cycle such that the erroneous value does not reach a downstream flip-flop by the end of the clock

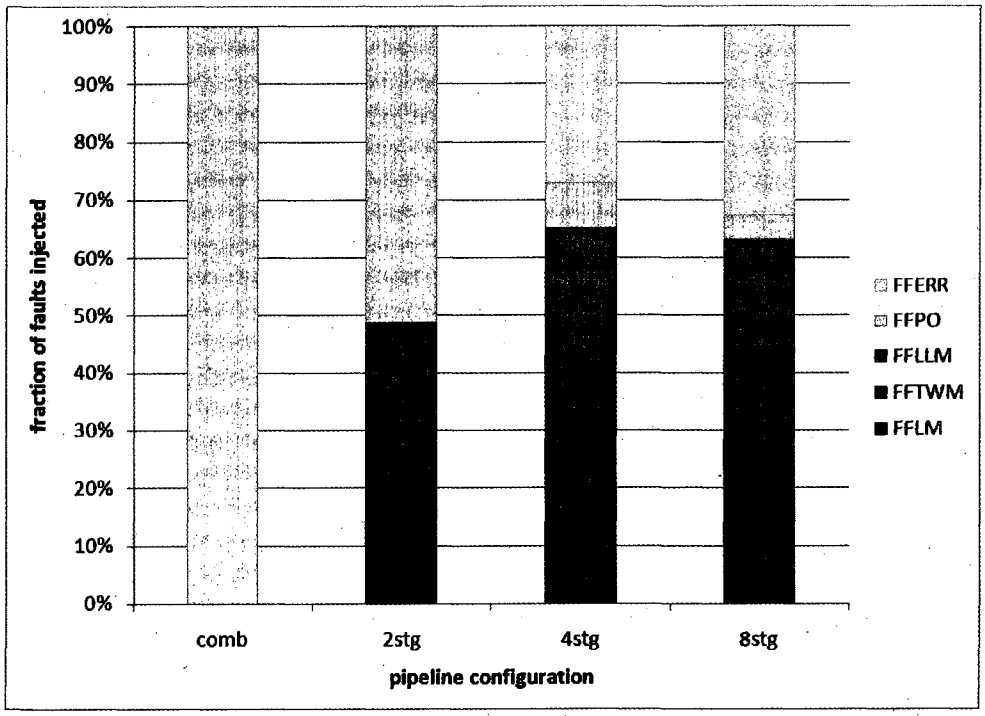


Figure 4.7: FP Multiplier Latch Fault Injection Breakdown.

cycle. In contrast to a SET, a fault injected directly into a flip-flop can be conceptually thought of as a stable value. Because of this, it significantly less likely that the erroneous value that propagates as a result of a direct strike on a flip-flop will be timing window masked.

In Figure 4.8, the overall latch error rates are plotted for both floating point units across all considered pipeline configurations. This error rate was measured experimentally by using the formula shown in Equation 4.4. For this calculation, all area values used are normalized to the combinational case (where the only

latches present in the circuit are for the primary outputs).

$$\text{latch error rate} = (\text{normalized area}) * \frac{\text{FFPO} + \text{FFERR}}{\text{faults injected}} \tag{4.4}$$

Looking at this plot, it is clear that the soft error rates (specifically for the multiplier unit) more than double when the pipeline depth is increased by a factor of 2. The adder circuit has results similar to what was predicted by the previously developed intuition, having a roughly 2X increase going from combinational to two, two to four, and four to eight pipeline stages. In contrast, there is a 4X increase observed in the measure latch soft error rate going from the two stage to four stage cases. This unexpected growth in the observed soft error rate can mainly be attributed to a larger than expected growth in latch count, as can be observed in Table 4.1.

The latch derating, shown as the rightmost part of Equation 4.4, is plotted in Figure 4.9. Both derating factors are 1 in the combinational case because all flip-flops belong to primary outputs, meaning that no masking can occur. The derating values plotted in this figure again depend on logical and timing window masking, but in contrast to the combinational logic experiment, logical masking is the dominant phenomena. The dominance of logical masking is evident in Figure 4.9 by the approximate equivalence of the measured derating values for the 2, 4, and 8 stage pipeline cases.

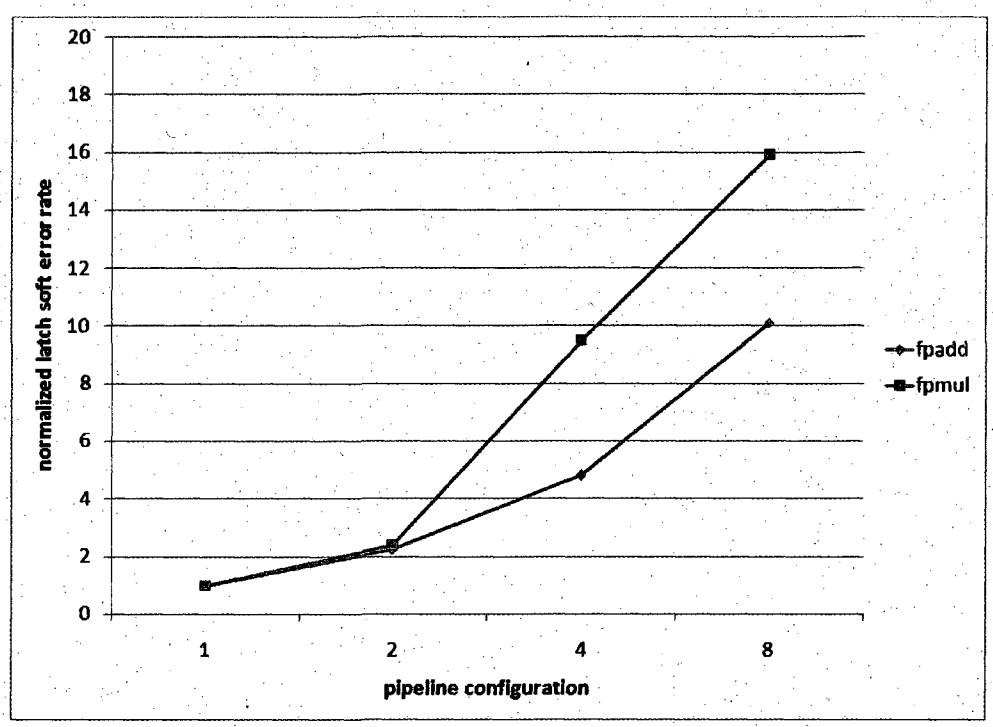


Figure 4.8: Measured Normalized Latch Soft Error Rate.

Relative Contribution of Latch and Logic Strikes

$$\text{Combined SER} = (\text{logic area} * \text{logic derating}) + (\text{latch area} * \text{latch derating}) \quad (4.5)$$

Because the floating point units evaluated in this study are vulnerable to particle strikes in latch storage as well as combinational logic gates, it is also interesting to look at the relative contribution of each type of fault to the overall failure rate. The expression displayed in Equation 4.5 was used to calculate the overall error rate.

The area values used for this calculation are all normalized to the combined

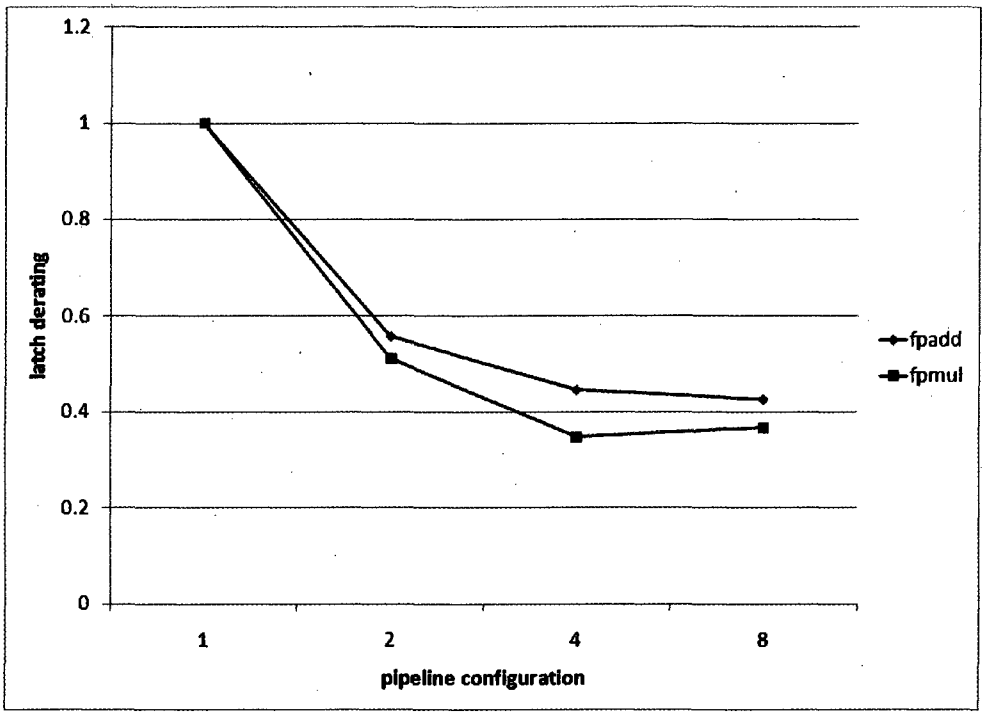


Figure 4.9: Measured Derating for Latch Strikes.

area of the baseline (combinational case). The combined error rate (with stacked bars representing the contribution of latch and logic error rates) is shown for the floating point adder in Figure 4.10 and for the multiplier in Figure 4.11. From these figures it is clear that particle strikes in combinational logic gates as well as flip-flops contribute significantly to the overall number of failures observed, and that both sources need to be dealt with in order to reduce the overall error rate. This is particularly true for the deeper pipeline depths, where flip-flops make up a larger part of the vulnerable area. In particular, the contribution of the latch soft error rate to the overall error rate rises significantly faster for the floating point adder. The reason for this dramatic increase can be explained by the rightmost

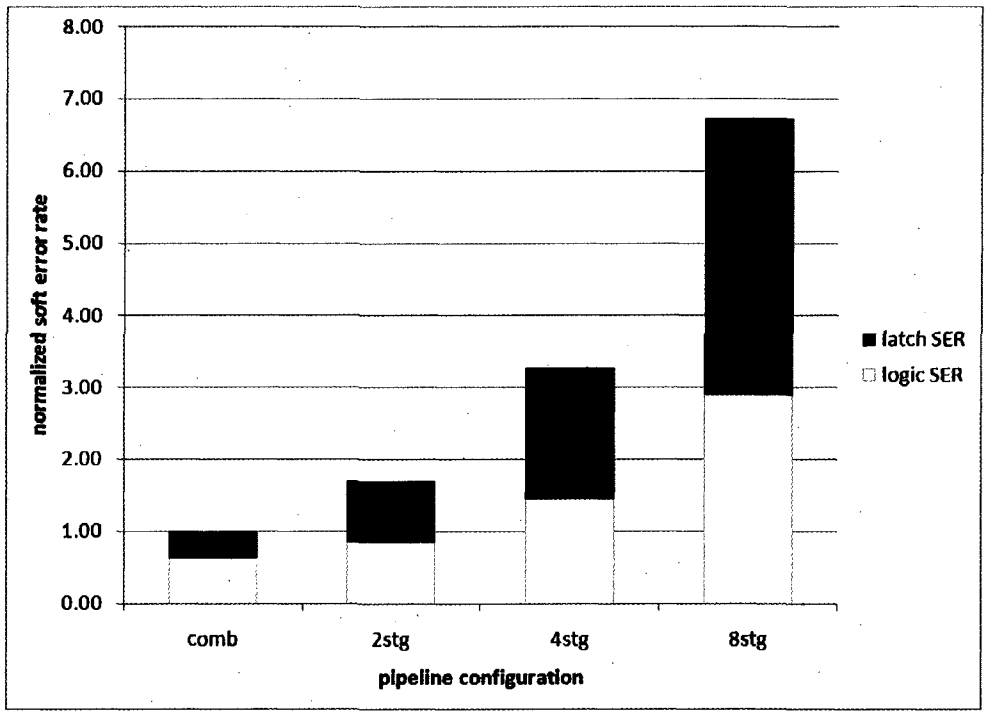


Figure 4.10: Combined Soft Error Rate for Floating Point Adder.

column of Table 4.1. There are less gates overall in the floating point adder, so flip-flops take up significantly more area.

4.3 Challenging the Conventional Intuition

While the results presented in the previous section do validate the previously developed intuition, the experiments conducted do not represent a fair comparison between the different pipeline configurations for each circuit. The reason that these experiments are not valid is they only represent of comparison of error rates. As was discussed in Chapter 2, system reliability is commonly quantified

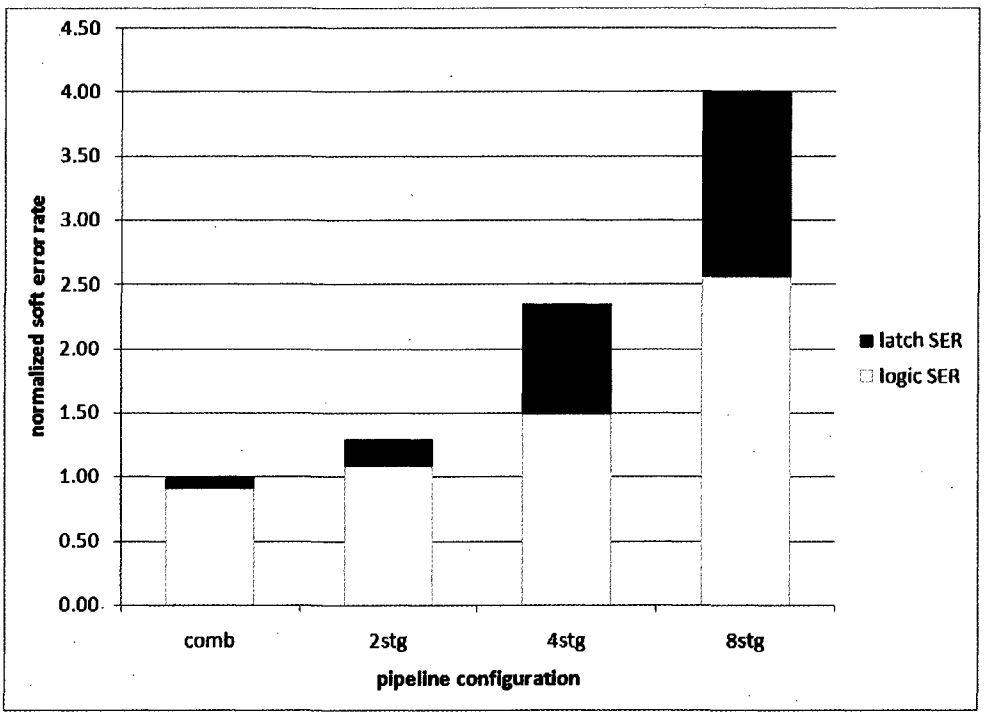


Figure 4.11: Combined Soft Error Rate for Floating Point Multiplier.

in terms of a error rate λ (and its reciprocal MTTF). Directly comparing error rates of two systems is only valid when both systems take the same amount of time to complete a task. Ultimately, the impact of errors on a system is represented by the product of the failure rate and the time time system is running, as shown by equation 4.6.

$$\text{errors observed} = \lambda * \text{time} \tag{4.6}$$

Despite the fact that MTTF is generally accepted as a standard reliability metric, it is not adequate for comparisons when two systems have different failure

rates and running times.

A Hypothetical Example

To illustrate how comparing error rates directly does not always result in a fair comparison, consider the two functional units shown in Table 4.3. This table shows two logic blocks which are identical in functionality. Additionally, Unit A is purely combinational, while Unit B is pipelined into 2 stages. The clock periods for Unit A and Unit B are set to 1 and 0.5 time units, respectively. Additionally, both units have the same amount of area devoted to logic, while Unit B has the twice the latch area. Because the two units are functionally the same, it is valid to make the assumption that approximately the same fraction of faults are logically masked by each circuit. Given this, the analytical expression for timing derating, shown in Equation 4.1 can be leveraged, allowing the combinational logic soft error rate to be approximated in this example as being proportional to the reciprocal of the clock period. In a similar manner, the latch error rate should be proportional to the latch area.

Scenarios Where Rate Comparison is not Appropriate

In the context of combinational logic SER, these assumptions imply that the error rate should be dependent only on the timing component of the derating factor. From this, it follows that the error rate of Unit B should be twice that of Unit A, due to its shorter clock period. If error rates are used as a direct

	Unit A	Unit B
Pipestages	1	2
Clock Period	1	0.5
Logic Area	1	1
Latch Area	1	2
Logic Error Rate	1	2
Latch Error Rate	1	2

Table 4.3: Hypothetical Functional Unit Descriptions. All values shown in terms of arbitrary units.

comparison here, the conclusion would be that Unit A is more reliable, since its longer clock period allows more particle strikes to be timing-window masked. Consider the scenarios shown in Figure 4.12. This figure is designed to illustrate the ways that each unit described in Table 4.3 could complete four arbitrary units of work over time. The top portion of this diagram depicts how these four units of work would be completed over time by Unit A. The lower portions of this figure depict two different ways Unit B could complete the same amount of work. These two cases represent scenarios where Unit B is not and is the execution bottleneck, respectively. In all cases, the x-axis represents time (in arbitrary time units), while the y-axis (for each of the three scenarios) represents the status of a particular pipe-stage. Colored and blank regions represent periods of time where a particular pipe-stage is computing or idle, respectively.

In the case where Unit A is processing this work, all four time units are needed, so the 1 pipe-stage in that particular circuit is always busy. In this case the expression 1*4T (it takes four time units for unit A to complete the work shown in Figure 4.12, at an error rate of 1) represents the total number of errors

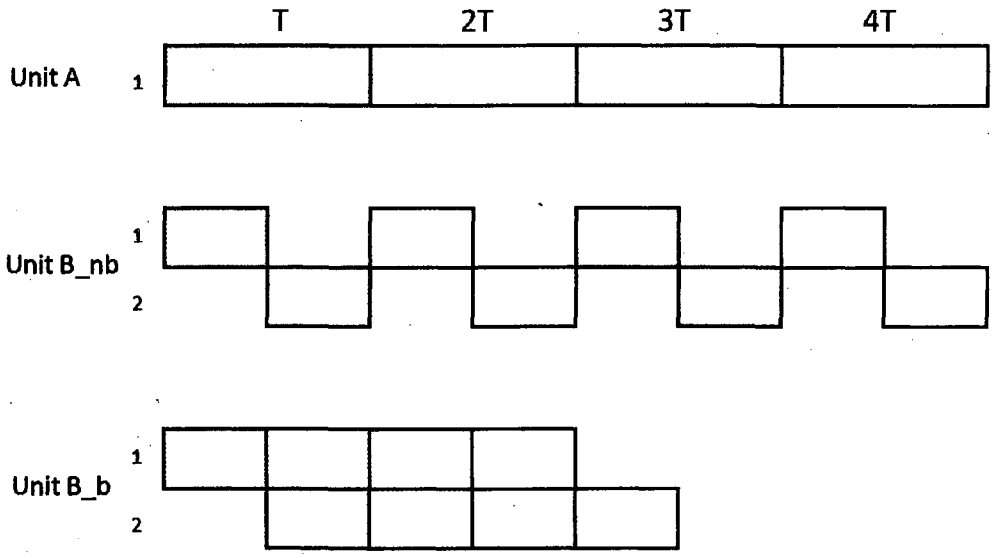


Figure 4.12: Timing Diagram for Instruction Processing.

that would be observed for Unit A. In contrast to this, consider the scenario shown in middle of Figure 4.12. In the case where Unit B is not the performance bottleneck units of work will arrive for processing at the same rate as the scenario shown for Unit A. In this case, despite the fact that it takes the same amount of time to complete the task on both units, Unit B will be idle half of the time. This is illustrated in Figure 4.12 by the unshaded regions in the timing diagram. Because particle strikes are uniformly distributed across time, only half of the strikes on Unit B will hit pipe-stages computing valid results. A similar situation occurs in the third scenario depicted, where Unit B is the system bottleneck, and units of work arrive and are processed as fast as possible. In both of these cases, the expression 2*2T represents the number of errors that would be observed (it takes two time units to complete the work, at an error rate of 2). To summarize,

when only rates are compared, Unit A will be chosen as the more reliable design, as it has an error rate of 1, while Unit B has an error rate of 2. In contrast, when the amount of errors observed is used as a comparison metric, 4T errors will be observed in both cases implying that Unit A and Unit B are equivalent in terms of reliability. A similar situation arises in the context of latch SER. The increase in error rate due to the area increase should also be offset by the shortened amount of time it takes to complete the assigned work.

Using the Right Metric

As was stated previously, MTTF is a widely used metric for reliability. It is typically calculated as the ratio

$$\text{MTTF} = \frac{\text{total time}}{\text{number of errors encountered}} \tag{4.7}$$

which simplifies to

$$\text{MTTF} = \frac{\text{total time}}{\lambda * \text{total time}} = \frac{1}{\lambda} \tag{4.8}$$

Comparisons using this metric have the implicit assumption that the total time required to perform computation in each system is identical. This is not the case for our functional unit comparison, meaning that MTTF is not the correct metric to use.

Weaver et. al more recently proposed an alternative reliability metric, mean

instructions to failure (MITF) [73]. MITF is calculated as the ratio

$$\text{MITF} = \frac{\text{instructions committed}}{\text{number of errors encountered}} \tag{4.9}$$

which simplifies to

$$\text{MITF} = \frac{\text{UWPC} * \text{total time} * \text{frequency}}{\lambda * \text{total time}} = \frac{\text{UWPC} * \text{frequency}}{\lambda} \tag{4.10}$$

The original work proposing MITF expressed the metric in terms of instructions per cycle (IPC), as the work was proposed in the context of considering the effects of soft errors in microprocessor. In the context of this discussion MITF is expressed in terms of units of work per cycle (UWPC), where a unit of work is described as the amount of work done in a single pipe-stage of Unit A or B. The implicit assumption made by this metric is that the default unit of work (an "instruction" in [73]) is consistent across all systems being compared. In our comparison, the unit of work is not consistent. One unit of work for Unit A is equivalent to two units of work for Unit B.

In order to accurately compare system with inconsistent units of work, Reis et. al proposed a more generalized metric, mean work to failure (MWTF) [53]. This metric was originally proposed to provide fair comparisons of reliability across dissimilar architectures, which might have inconsistently defined units of work.

MWTF is defined as the ratio

$$\text{MWTF} = \frac{\text{amount of work completed}}{\text{number of errors encountered}} \tag{4.11}$$

which simplifies to

$$\text{MWTF} = \frac{\text{amount of work completed}}{\lambda * \text{execution time}} \tag{4.12}$$

This metric takes a more abstract definition of what constitutes a unit of work. It also factors in the difference in execution time for different systems. For this metric, typically something larger (like a transaction or an entire benchmark) is used as the basis for a unit of work. This broader definition allows for consistency across systems that may be very different. For the purposes of our comparison, it is best to define a unit of work, as one item processed by Unit A. This means that (looking at the diagrams in 4.12 and 4.12) that both Unit A and Unit B are doing 4 units of work (even though Unit B completes the work in 8 clock cycles rather than 4). Applying this metric, the MWTF for Unit A would be

$$\frac{4}{\lambda * 4T} = \frac{1}{T} \tag{4.13}$$

and MWTF for Unit B would be

$$\frac{4}{2 * \lambda * 2T} = \frac{1}{T} \tag{4.14}$$

Which is the result we expected from the discussion in Section 4.3.

Revised Intuition

The application of the appropriate metric, MWTF, to evaluate the effects of pipelining a functional unit make it clear that to a first order an increase in pipeline depth should have no effect on the SER. Several experiments were conducted to validate this revised intuition.

4.4 Fair Analysis

Combinational Logic SER

In order to further explore the revised intuition developed in the previous section, the results from the experiments described in Section 4.2 were adjusted to account for the execution time differences that would exist between pipeline configurations. The adjusted results are shown in Figure 4.13. From this figure it is clear that the results of this experiment do not match our revised intuition. Instead of staying constant, the measured derating actually decreases across pipelining configurations, when adjusted for execution time. The propagation of SETs to multiple flip-flops along unbalanced paths is responsible for this counterintuitive result, and is only observable when the effects of timing window masking are modeled explicitly.

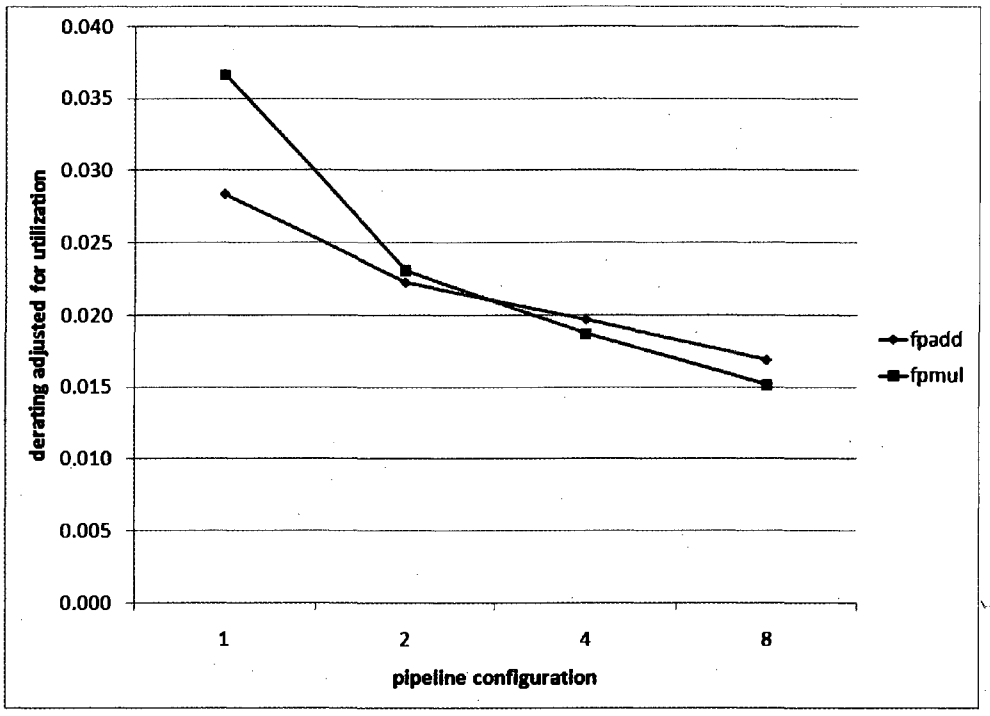


Figure 4.13: Plot of Logic Derating Adjusted for Execution Time.

SET Fanout Effects

A diagram describing the second order effect responsible for the counterintuitive results in Figure 4.13 is shown in Figure 4.14. In the scenario depicted in Figure 4.14, transient pulses fan out from a single combinational logic gate to two downstream flip-flops. In this case, the path length and delay from the gate to each flip-flop is different. In this situation, the absolute window of time where at least one flip-flop could be corrupted is lengthened. This new window of time was defined as the effective SET width. This quantity is illustrated in Figure 4.14 as the superposition of the SETs arriving at each flip-flop. The equation shown in

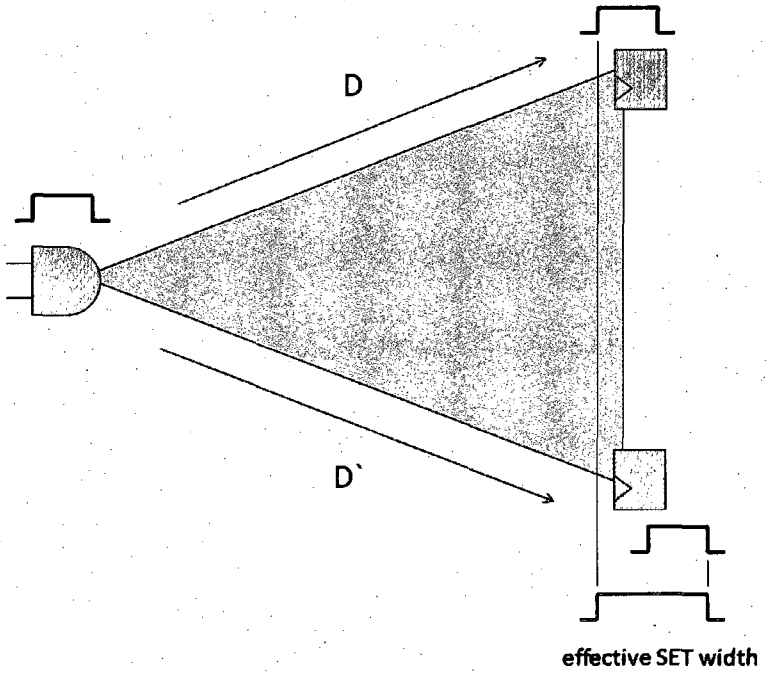


Figure 4.14: Illustration of SET Fanning out to Multiple Flip-flops.

Equation 4.1 to characterize timing derating can be written as shown in Equation 4.15 to account for the impact of this second order effect.

$$T_{derating} = \frac{\text{effective SET width - w}}{C} \quad (4.15)$$

The average effective SET width observed during several fault injection experiments was measured and plotted for different pipeline configurations in Figure 4.15. As was discussed in Chapter 3, the average transient width of an injected SET is 100 ps. From this figure it is clear that at the shallower pipeline depths, the SET fanout effect is significantly more pronounced. The effect is more pronounced in these pipeline configurations because there is the same amount of

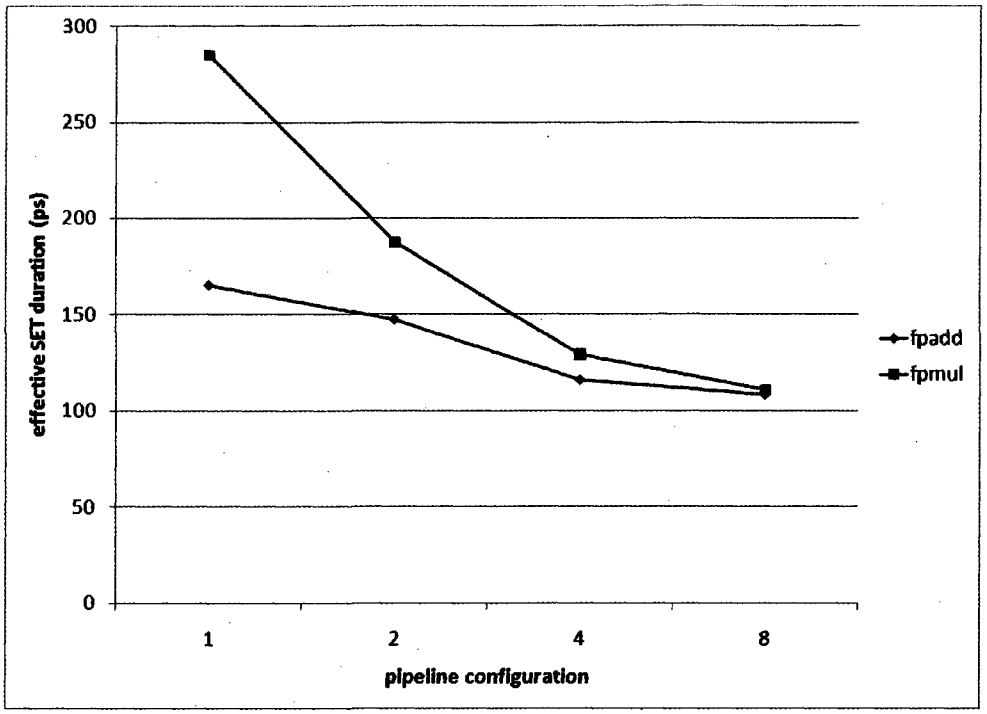


Figure 4.15: Experimental Measurement of Effective SET Width.

combinational logic, but fewer flip-flops, implying that a transient will have to propagate through more levels of logic (and thus fanout further) before reaching a flip-flop.

If the SET fanout phenomena mentioned previously is more pronounced at shallower pipeline depths, one would also expect that instances of multiple bit flips (cases where the injection of a SET results in multiple flip-flops capturing wrong values) would also be increased. In Figure 4.16, histograms displaying the number of bit flips caused by a single SET are shown for both the floating point multiplier and adder. These histograms show that the number of multiple bit flips does indeed rise proportionally with the effective SET width. The bar

corresponding to the 2 stage floating point multiplier represents an interesting case in Figure 4.16. Intuitively, this circuit would be expected to have fewer multiple bit flips than the combinational case, as SETs only need to propagate through half the levels of logic to reach a flip-flop (and thus have less opportunity to fanout). Surprisingly, the histogram in Figure 4.16 actually shows an increase in the amount of multiple bit flips for the 2 stage case. This unexpected behavior is an artifact of how the circuit was automatically pipelined by the Synopsys Design Compiler tool chain. While the combinational multiplier has 83 flip-flops at the outputs of the circuit, the 2 stage pipeline version has 306 additional flip-flops separating the two pipelined stages. This means that in the combinational case, many SETs are fanning out in the middle of the circuit and then fanning back in as they propagate to the outputs, while in the in two stage version only the fanning out is occurring.

It should also be noted that the SET fanout phenomena discussed in this work was also discovered concurrently by [24][1]. Their work was performed in the context of selecting gates for hardening. In this study each gate was assigned a window determined through static timing analysis, which represented the period during the clock cycle where a particle strike could occur and arrive at the input of a downstream flip-flop during the latching window. In this work the observation was made that some gates had significantly larger windows than others because of unbalanced fanout to flip-flops along multiple paths.

Despite the fact that this study was conducted entirely in the context of the

[1]The results in this chapter were first collected in 2007.

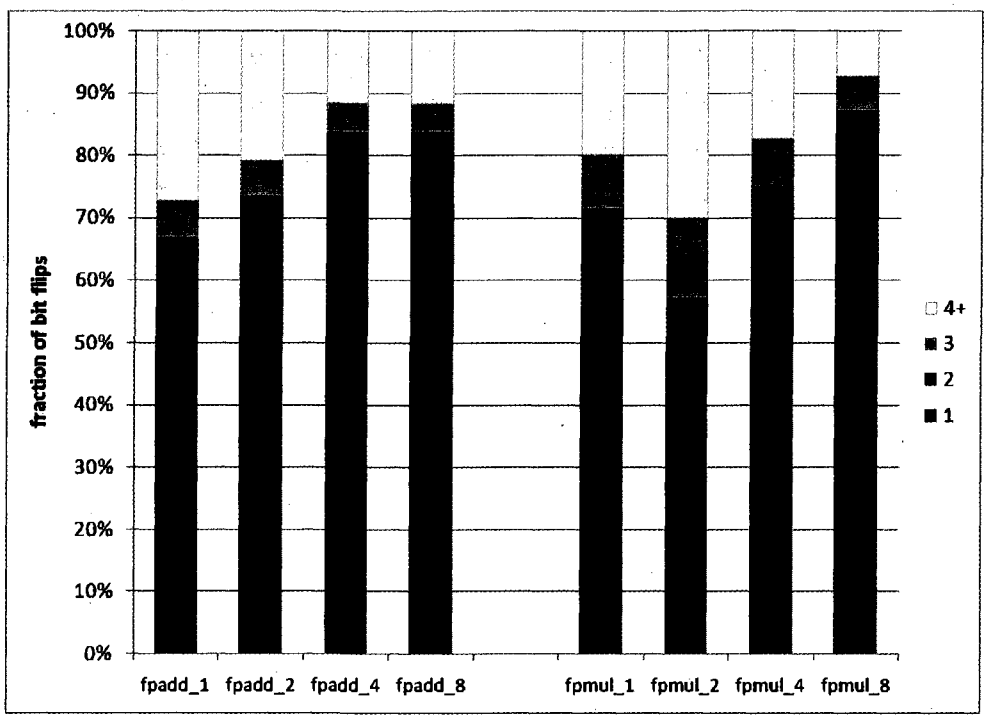


Figure 4.16: Histogram of Number of Bits Flipped by a SET.

65nm technology generation, which is what the circuit level modeling was based on, the SET phenomena uncovered in this study should persist even with technology scaling. The effective SET width term in Equation 4.15 will continue to dominate the timing derating with scaling, because SET width will continue to increase as feature sizes continue to shrink.

Combined SER

The combined SER adjusted for execution time (essentially the quantity expressed by Equation 4.6) is plotted in Figure 4.17. The execution times are normalized to

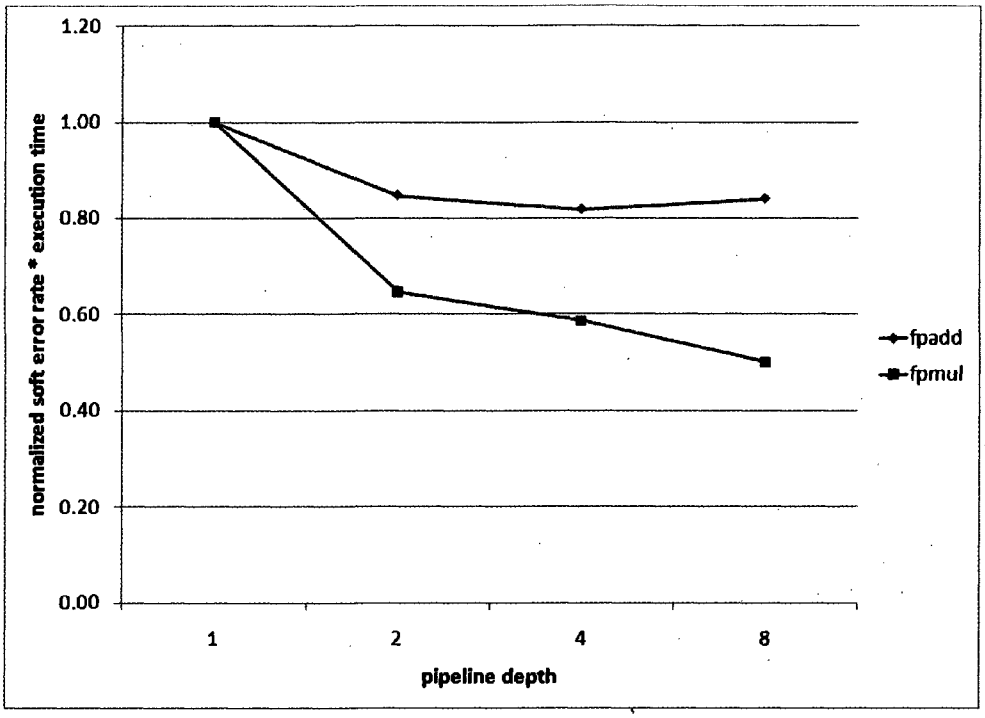


Figure 4.17: Combined SER Adjusted for Execution Time.

the combinational case. Because the majority of vulnerable area in all comparison points shown can be attributed to combinational logic gates, the effects described in the previous section have a profound effect on the scaling of the soft error rate. This is especially true for the shallower pipeline depths, where not only a larger fraction of area can be attributed to logic, but also SET fanout affects are reducing the amount of timing window masking. For the deeper pipeline depths, increasing latch area means the latch soft error rate has a larger influence on the overall error rate observed.

4.5 Conclusion

In this chapter, the effect of pipelining logic, commonly cited as a reason the logic soft error problem is being exacerbated, was explored. In this exploration, the fallacy in this line of thinking (the use of MTTF as a comparison metric) was uncovered, and the correct metric to use, MWTF, was identified. The newer MWTF metric was used to refine the previously cited conventional intuition to having the soft error vulnerability of a circuit be independent of pipeline depth, rather than directly proportional to it. In validating this revised intuition, a second order effect causing adjusted failure rates to decrease at deeper pipeline depths was uncovered. The use of the correct comparison metric (MWTF), along with this second order effect, SET fanout, mean that deeper pipelines are in many cases more resilient that their shallower counterparts, reversing the previously held intuition.

Chapter 5

Choosing the Right Strategy for Protection

5.1 Introduction

Up until this point, this dissertation has focused on refining both the intuition and the methodology used to study transient faults in logic. This chapter goes in a more practical direction, identifying appropriate mitigation techniques for a given logic block. In this study, taxonomies are developed to classify techniques according to the level and means by which faults are handled, and to classify logic blocks according to functionality and gate level structure. Related to this logic block classification, a case study is performed, in which the soft error vulnerability of a z80 parallel decoder is characterized in detail. This case study serves the purpose of illustrating the structure of the resulting artifacts produced by

transient faults at the output of a logic block. The experimental results presented in this study show that these artifacts cannot always be modeled as SEUs, which is an assumption commonly made by studies on reliability performed at higher levels of abstraction [43][12][49]. Finally, this chapter concludes by qualitatively determining which class of mitigation technique is most appropriate for each different type of logic block.

5.2 Classification of Protection Techniques

Macro-Level Replication Techniques

Mitigation techniques in this category generally involve replicating some significantly large portion of hardware (or in some cases software) either spatially or temporally, and comparing the results of computation in order to detect the presence of faults. In some cases, this replication can be global, at the system level [8], processor level [62][6][7], or at the thread level [34]. In other cases, replication can occur only on a functional unit level[48], or even in software[52]. While techniques in this category typically exhibit a significant amount of error tolerance due to the large amount of hardware (or software) replicated, the overhead incurred in terms of power, area, and performance are often large. These high costs (specifically for the global subset of this category) have inspired many other research proposals with the expressed goal of achieving some form of replication, while reducing the overheads incurred [63][66][16][7]. The ideas proposed in this thesis are primarily intended for commodity computer systems, where the

costs of techniques in this category would be prohibitive.

Property Based Checking Techniques

In contrast to techniques which provide error tolerance through macro-level replication, property checking approaches detect and/or correct errors by verifying a chosen property of either the result computed (in the case of a logic block) or a value held in storage (in the case of a memory cell). In some cases whether or not the property is verified allows an absolute conclusion to be drawn on whether or not an error occurred. Information redundancy techniques used for storage fall into this category, as well as code-based checking approaches used for logic blocks [3][72]. Additionally, techniques which check correctness [51][50][60] also fall into this category. In other cases, a property check failing only indicates a guess that an error occurred, meaning that there is a potential for performance-degrading false positives. Symptom-based approaches [71][43] fall into this category.

Implementation-Level Techniques

Mitigation techniques in this category generally involve manipulating individual gates and/or transistors in order to improve soft error tolerance. As a consequence of this, these techniques often are not integrated into a design until late in the development process. A positive aspect of this situation is that in many ways less design effort may be required, because a reliability solutions in this category

typically do not impact the functional verification of a design (as is commonly the case with techniques in the first two categories). Because techniques in this category generally manipulate low level components, some effort is needed to only modify a subset of components in order to ensure that other design goals are not sacrificed for the sake of reliability. For this reason, techniques in this category are often coupled with heuristics in order to make these decisions.

Techniques in this category can provide reliability by manipulating components in several ways. Many of the early implementation-level techniques provided reliability by resizing individual transistors within a circuit, increasing the drain capacitance (and thus minimum amount of charge (Q_{crit}) needed to be generated by a striking particle in order to induce a fault)[81]. This subset of techniques can be applied to transistors inside of combinational logic gates as well as in storage cells. Another subset of techniques also exists in which the presence of transient activity is detected at the input of sequential elements [14][38][31]. Many of these techniques were originally proposed to enable varying degrees of timing speculation, where logic circuits are clocked at a frequency higher than what their critical paths would safely allow. These techniques are useful for SET detection because timing violations manifest themselves in the same manner as SETs arriving at flip-flop inputs.

In a correctly designed synchronous logic block, the data input lines of the flip-flops at the output of the circuit should have stable values at the end of a clock cycle (when the rising edge occurs). In the case of a timing violation, when the rising clock edge occurs, a data input line still might not have settled to its

final value and may continue to change after the rising edge. Given this, the most straightforward way to detect a timing violation is to check the value of a data input line during the period of time after the rising edge of the clock, but before the work from the previous pipe stage can propagate through the circuit. If there is no change in the value of the data input line during this period, then the value was stable and no timing violation occurred. However, if there was a detected change, then there is a timing violation.

SETs manifest themselves in a similar manner. Recall from the discussion in Chapter 2 that a SET flips the value stored by a downstream flip-flop when an erroneous value is present at the data input line of the flip-flop during the latching window, or the period around the rising edge of the clock. By monitoring the data input to a flip-flop during and after the latching window, the presence of SETs can also be detected.

Techniques in each category (or combinations of categories) can be used to represent the entire possible solution space of a fault tolerant computer system. At one extreme end, a macro-level approach similar to the NonStop system provides fault tolerance by replicating every component [8]. At the other end, property-based and implementation level techniques can be combined in order to create a microprocessor that is fault tolerant with a minimum amount of replication.

5.3 Classification of Logic Blocks

Besides classifying mitigation techniques, it is also useful to categorize logic-dominated structures within a system according to structure and functionality. With respect to these attributes, logic blocks are categorized as either datapath blocks, hybrid blocks, or control blocks. Each category has distinct defining characteristics with regard to how transient faults propagate. These characteristics will be considered in Section 5.4 when different strategies for protection are discussed.

Datapath Blocks

The Datapath category is primarily intended to include logic blocks with minimal control logic and simple functionality. In the context of a conventional microprocessor the most prominent location of datapath logic would be in the execute stage of the processor pipeline, where ALUs perform simple arithmetic and logical operations on instruction source operands. In addition to the execute stage, similar logic structures typically are spread throughout a system. In many cases these logic blocks are trivial (a single level of gates can be used to perform a logic operation), and regular in structure. This implies that logic blocks of this type would not be very amenable to logical masking of transient faults. In general, datapath blocks blocks are important to protect, as these logic blocks are responsible for generating values used to drive control and memory data flow, either directly or indirectly via register file communication.

Hybrid Blocks

This category is meant to include logic blocks which contain significant amounts of both datapath and control logic. Floating point units typically have a large amount of datapath logic to calculate the fractional portion of a floating point result. Also a floating point unit is generally capable of performing the required computation for multiple instructions, usually in both single and double precision modes. As an example, the floating point adder studied in Chapter 4 can perform single and double precision addition, subtraction, and comparison, as well as conversions to and from integers [29]. A unit like the one just described needs a significant amount of control logic in order to choose the appropriate functionality to utilize in order to execute an instruction. For blocks in this category, transient faults in datapath components can propagate in a manner similar to the scenarios described for the previous category, while strikes in control logic can result in more serious errors, as the wrong function can be computed entirely. Fortunately, control logic tends to be less regular in structure than datapath logic, implying that transient faults in this case are more likely to be logically masked.

Control Blocks

This final category includes logic blocks containing only control logic. This category is meant for units that compute state that is only used to control other units, and not for the direct computation of values stored in the register file or memory. The most prominent examples of blocks falling into this category

would be instruction decoders, finite state machine logic, and on-chip structures for environment monitoring (such as a power management unit). Units in this category will typically logically mask a significant amount of transient faults, but the cases where faults are not logically masked can potentially have catastrophic effects on the rest of the system. For example, a hypothetical scenario could occur where a transient fault during instruction decode could invert the condition used to determine whether a branch instruction is to be taken (a branch if equal could be decoded as a branch if not equal). In this scenario, control flow would be forced down the wrong path. In some cases the end effect of this situation could be severe (a silent data corruption event), moderate (a crash), or benign (no visible difference in execution, as studied by [69]).

Case Study: z80 Instruction Decoder

In order to better understand how faults propagate through control blocks, a parallel instruction decoder was designed and then characterized in terms of soft error vulnerability. Decoder logic was chosen as a particular case to study because instruction decoders are primarily composed of combinational logic and highly utilized. Also, recent work aimed at either modeling or mitigating faults within decoders has not considered their gate level structure [12][49][28]. Architectures featuring instructions with variable lengths are of particular interest, mainly because of the difficulties associated with decoding multiple instructions in parallel, as noted by [40].

The decoder implemented for this work decodes z80 instructions. The z80

[prefix byte], opcode, [displacement], [immediate data]
- OR -
two prefix bytes, displacement, opcode

Figure 5.1: z80 Instruction Format. Adapted from [58].

architecture is commonly used for embedded micro-controllers, and is thus substantially simpler (in terms of ISA complexity) than architectures prevalent in the high performance general purpose microprocessor design space. While z80 is substantially simpler than x86, several commonalities do exist between the two ISAs. First, both x86 and z80 have variable length instructions [40][57]. Second, z80 was originally designed to be binary compatible with the 8080 ISA, an predecessor to x86 [57]. In addition to this, this decoder implementation is inspired by industry published descriptions of parallel decoder designs[41], mapping a single z80 instruction to potentially many RISC operations. For these reasons, I am confident that the decoder implemented for this study has a similar functional structure (and thus similar transient fault propagation characteristics) to a parallel x86 instruction decoder within a high performance microprocessor.

Instructions in the z80 instruction set can be between one and four bytes in length, and and are formatted in one of the two forms shown in Figure 5.1. In the case of the first form shown, the prefix byte, displacement byte, and immediate data field are optional depending on the instruction being represented.

Decoder Design

A block diagram of the parallel z80 decoder is shown in Figure 5.2. The decoder is capable of decoding up to three z80 instructions per cycle, and each instruction can be translated into up to 12 RISC operations. The decoder is pipelined into 3 stages: speculative length decode, opcode identification, and translation. The boundaries between each of the stages are denoted by the dotted lines in Figure 5.2. Decoding multiple instructions in parallel can be difficult for ISAs featuring variable instruction lengths. The most difficult aspect of parallel decode in these cases is identifying the start of each new instruction following the first instruction, because the starting point of each instruction is dependent on the length of its predecessors. For example, the decoder implemented in this study takes four bytes of input, and is capable of decoding up to 3 instructions in parallel. While the first instruction can only start at the beginning of the first byte, the second instruction can start at the second byte, the third byte, or the fourth byte, depending on the whether the first instruction is one, two, or three bytes in length, respectively. The third instruction can either start at the beginning of the third byte (in the case where both preceding instructions are of length one), or at the beginning of the fourth byte (when one preceding instruction is one byte, and the other is two bytes in length). This means that for three possible instruction slots, there are six possible starting points (one for the first instruction, three for the second instruction, and two for the third instruction). Enumerating all possible starting points for this simple design should make it clear that realizing wide parallel decoders can be extremely difficult, especially for more complex ISAs such as

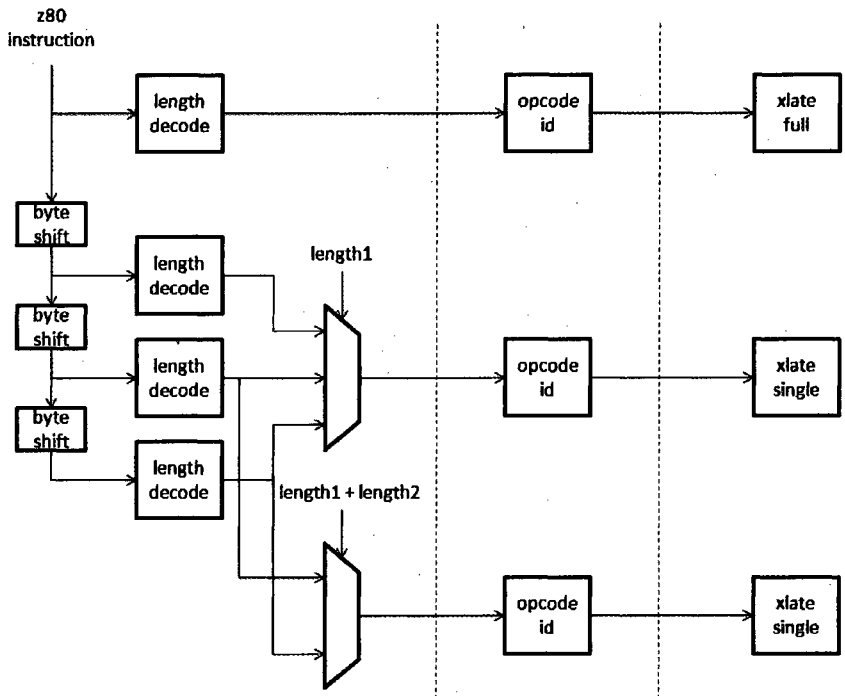


Figure 5.2: z80 Decoder Block Diagram.

x86, where instructions can have a larger range of possible lengths.

A naive approach for decoding variable length instructions in parallel would be to speculatively perform all decoding tasks from all possible starting positions for each instruction slot, and then select the correct decoded instructions to used based on the results of the first instruction (for the second slot), and the first and second instruction (for the third slot). While this would be a valid solution to the problem, it would require a significant amount of wasted computation, as six candidate instructions would be speculative decoded for just three slots in the best case. To circumvent this overhead, conventional parallel decoders adopt a strategy where the starting point of each instruction slot is determined

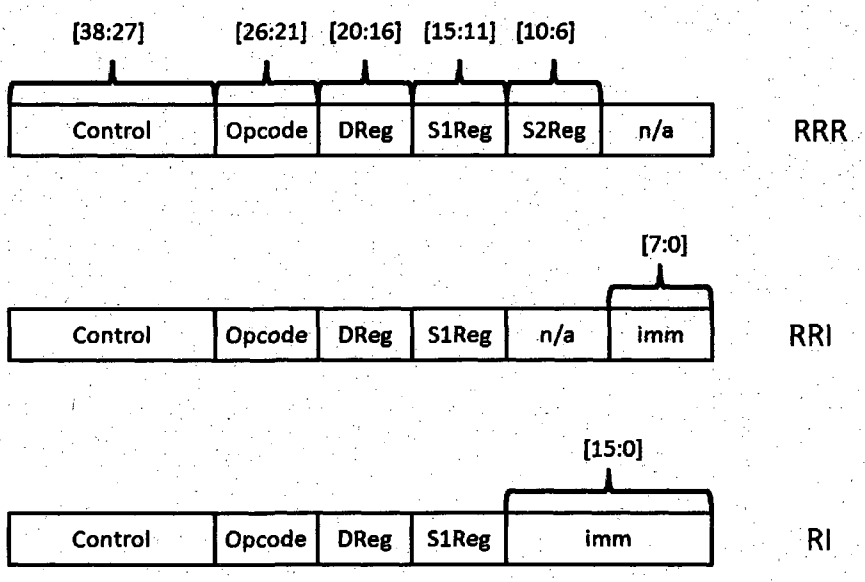


Figure 5.3: RISC Operation Format.

first (a process denoted as length decoding), allowing the other decoding tasks to be performed in a non-speculative manner. Dividing decode up in the manner described is advantageous in that the only speculative logic needed is in the length decode stage. It should be noted that while there are six starting points for z80 instructions as discussed previously, in Figure 5.2 there are only four length decoders present in the first stage of logic. The reason for this is that the results of speculative length decode starting from the third and fourth bytes can be used for instructions potentially in the second and third slots. The hardware for length decoding is relatively simple (as there are only four possible instruction lengths) and is thus an attractive candidate for speculation.

In the final two stages of decode, opcodes are first identified and then translated

into RISC operations. In the design implemented for this study, a RISC operation can have one of three forms depending on the nature of its source operands. The bit level format of each RISC operation form is shown in Figure 5.3. Each operation is 39 bits in length, and classified as either RRR (both source operands come from registers), RRI (one register source operand, one immediate source operand), or RI (a single immediate source operand). Inspired by the decoder implementation described in [41], the translation logic for the first instruction slot (labeled as xlate full in Figure 5.2) can translate any instruction, while the translate logic for the other two slots (denoted as xlate single) can only handle z80 instructions which are translated into a single RISC operation. This means that the decoder can produce up to 14 RISC operations per cycle. The decoder also includes three valid bits for reach instruction slot, as well as 14 valid bits for each possible RISC operation produced.

Soft Error Characterization

In order to understand how transients propagate through the decoder, characterization experiments were performed using the fault injection framework described in Chapter 3. In the first experiment, 30,000 faults were injected into combinational logic gates exclusively. The results of this experiment are shown in Figure 5.4.

Each slice of the pie chart shown in this figure represents the fraction of injected faults which result in a particular outcome. All of the possible outcomes that can occur when a fault is injected into combinational logic are shown in

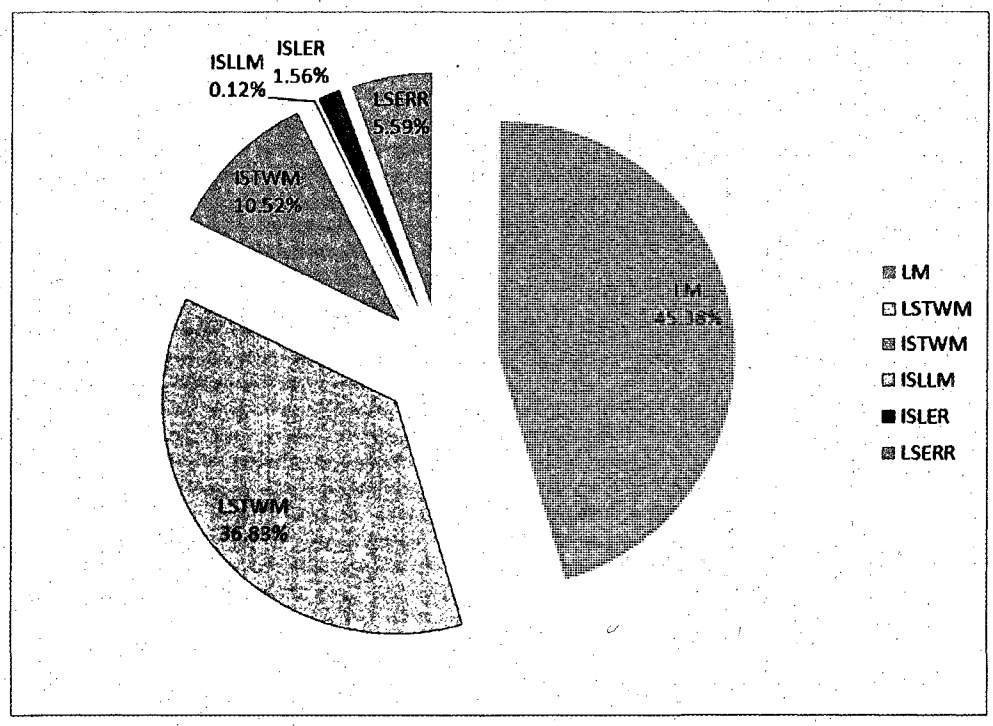


Figure 5.4: z80 Logic Fault Outcome Breakdown.

Table 5.1. Of all the faults injected into combinational logic gates, only 7.1 % (representing the combination of the ISLER and LSERR slices) result in errors. Probably the most interesting attribute of this particular circuit is its logical masking behavior. Looking at Figure 5.4, 45.4 % of the injected faults result in the LM outcome. This result is particularly interesting when it is contrasted with the small fraction of faults resulting in the ISLLM outcome. This outcome only occurs 0.12 % of the time overall, or 18 % of the time when a SET flips at least one bit (either the ISLER or ISLLM outcome).

The cause of this distinctly different observed logical masking behavior stems

Outcome	Description
LSERR	SET corrupts primary output flip-flop
LSTWM	SET reaches primary output flip-flop, but is logically masked
ISLER	SET corrupts intermediate flip-flop, error propagates to primary output
ISLLM	SET corrupts intermediate flip-flop, but error is logically masked
ISTWM	SET reaches intermediate flip-flop, but is timing window masked
LM	SET is logically masked before reaching flip-flop

Table 5.1: Possible Outcomes for Combinational Logic Transient Fault Injection.

from the functional structure of the decoder. As stated previously, the decoder logic is partitioned into length decode, opcode identification, and translate stages. In this design, many outputs of each internal stage directly influence the functionality of the succeeding stage. For example, opcode identification is performed in the second pipeline stage by looking at the opcode byte (whose position is determined by length decoding) as well as the instruction length (not necessary, but used to narrow the space of candidate opcodes). Because of this implementation, bit flips resulting from particle strikes in the length decode stage will likely result in incorrect opcodes being identified, and thus incorrect RISC operations being generated at the output of the decoder. A similar situation occurs in the translation stage. Any corruption in the opcode identifiers generated at the output of the second stage is likely to cause an error in translation. If the decoder was pipelined into a larger number of stages, the fraction of faults resulting in ISLLM outcomes (where the fault is masked after flipping a state bit) observed should intuitively increase. This is not the case in our design, because the ranks of flip-flops were placed at coarse logical boundaries. The observed logical derating is 54%.

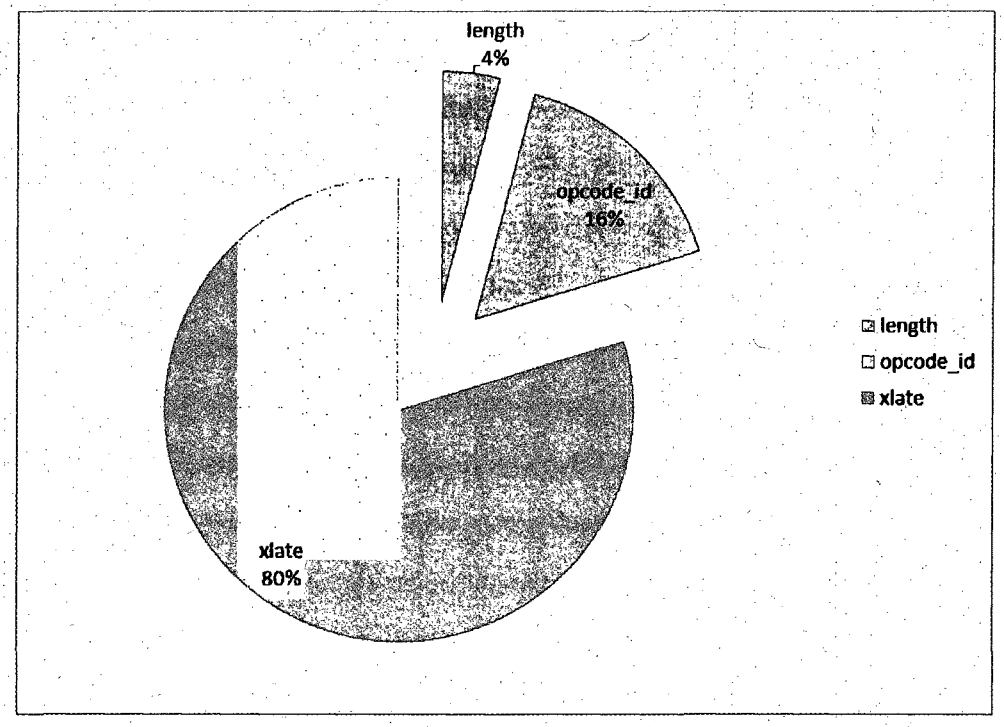


Figure 5.5: z80 Error Origin.

In Figure 5.5, the errors observed during fault injection are broken down according to the pipeline stage the fault originated in. The pie chart shown in this figure has 3 slices: length (representing the length decode stage), opcode_id (representing the opcode identification stage), and xlate (representing the translation stage). From this chart, it is clear that the majority of errors observed originate in the translation stage of the decoder. This is the largest stage of the pipeline in terms of area, so statistically a larger fraction of faults are injected into its gates.

Another interesting phenomenon explored in this case study was the sensitivity of decoder output bits to particle strikes in combinational logic. One of the original stated motivations for this chapter was to explore the choice made by

many studies at higher abstraction levels to always model the artifacts of particle strikes as SEUs. This choice has two implicit assumptions: first, all output bits are equally likely to be corrupted, and second, that only a single bit is corrupted at a time.

In order to examine the validity of the first assumption the derating per output bit, calculated by dividing the number of times each output bit was corrupted by the total number of faults injected, was measured and displayed in the scatter plot shown in Figure 5.6. Each point in this plot represents the probability that a particular output bit's value will be incorrect due to a transient fault. The y-axis in figure represents the derating per bit of each output while the x-axis is the number assigned to that particular node. The netlist format used for the developed tool chain assigns each input, gate, flip-flop, and output within a circuit a unique number. The range of output bits corresponding to each RISC operation slot is shown in Table 5.2. Looking at this plot, it is clear that some output bits are significantly more likely to be corrupted than others.

As was stated previously, the implemented decoder design is capable of translating the instruction in the first slot into up to 12 RISC operations, and the instructions in the second and third slots into a single RISC operation. The majority of the instructions in the trace used as input stimulus during in this experiment had short translations, which why the bits on the left hand side of the graph (which correspond to the translation RISC operations for the first instruction slot) and the bits on the extreme right hand side of the graph (which correspond to RISC operations for the second and third instruction slot) have higher derating values.

Bit Number	Description
454 - 441	RISC operation valid bits
493 - 455	RISC operation 1
532 - 494	RISC operation 2
571 - 533	RISC operation 3
610 - 572	RISC operation 4
649 - 611	RISC operation 5
688 - 650	RISC operation 6
727 - 689	RISC operation 7
766 - 728	RISC operation 8
805 - 767	RISC operation 9
844 - 768	RISC operation 10
883 - 845	RISC operation 11
922 - 846	RISC operation 12
961 - 923	RISC operation 13
1000 - 962	RISC operation 14
1003 - 1001	z80 instruction valid bits

Table 5.2: Description of z80 Decoder Output Bits.

In addition to looking at which output bits were most likely to be corrupted, the number of output bits corrupted simultaneously by a single injected fault was also studied. The pie chart shown in Figure 5.7 shows a breakdown of the observed errors in the decoder classified by fault origin (xlate, opcode_id, length) and how many bits were simultaneously corrupted (s - single, m - multiple). Looking at Figure 5.7, nearly 60% of the injected faults result in a single output being incorrect, with a vast majority of those cases originating in the translate stage. This case is denoted by the slice labeled xlate_s in the pie chart. Apart from this, multi-bit errors comprise 40% of the overall errors observed, including the majority of cases originating from the length decode and opcode identification

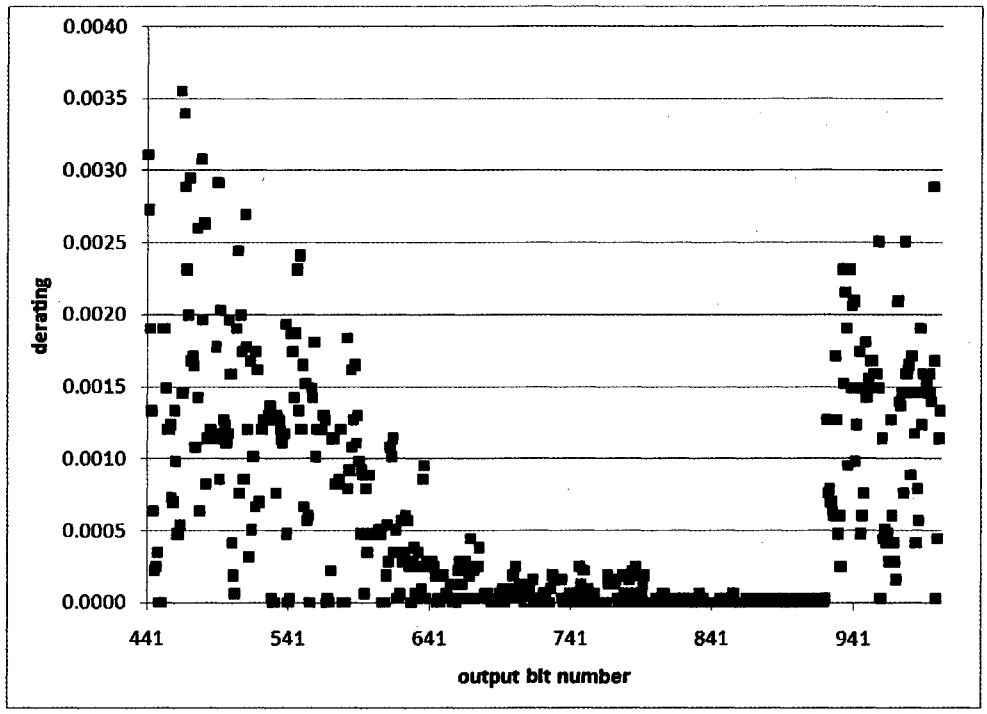


Figure 5.6: z80 Decoder Output Derating per Bit.

stages.

The histogram shown in Figure 5.8 represents a breakdown of how many output bits are incorrect in each multi-bit error case. The individual cases where different numbers of bits flip are represented on the x-axis, while the stacked bars indicate the fraction of total multi-bit output errors in that case originating from a particular pipeline stage. From this histogram it is clear that the cases where a higher number of bits are corrupted are more likely to originate from particle strikes in the length decode and opcode identification stages.

Finally, the design of the decoder itself has significant implications with respect to how detectable errors are at program outputs. The majority of conven-

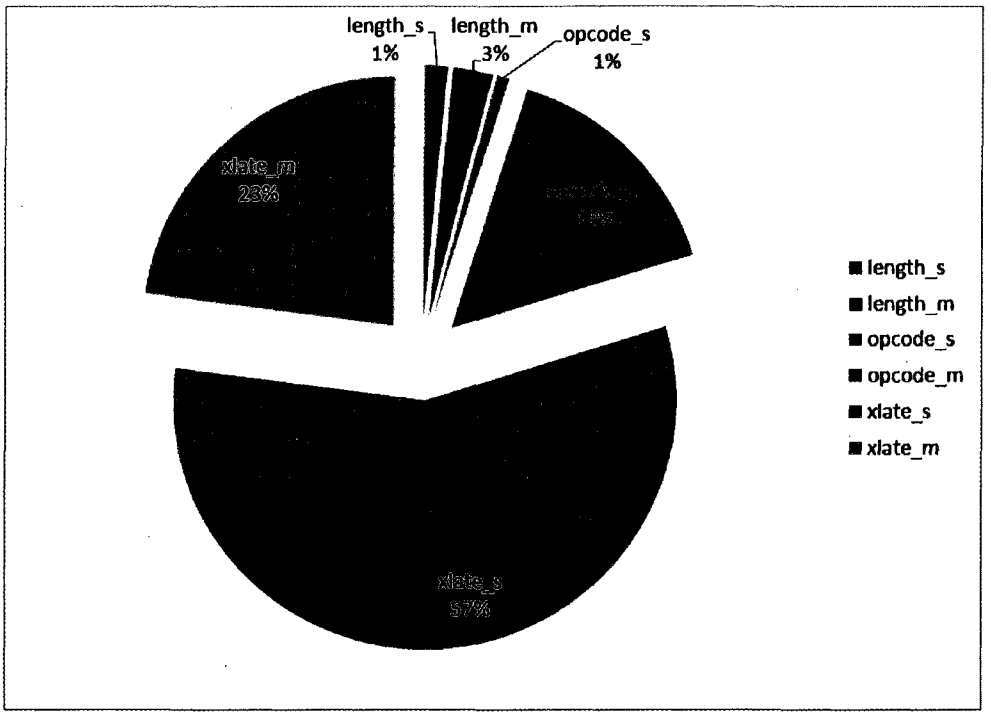


Figure 5.7: z80 Output Error Characterization.

tional CISC architectures translate instructions into RISC primitives in order to simplify the design of the execution hardware. One general attribute of RISC ISAs is their regularity, specifically in terms of which combinations of opcodes, source registers, and destination register can be combined to form valid instructions. From an error detection standpoint, an irregular ISA could be attractive because if many invalid opcode-register combinations exist, it is likely that a transient fault in the decoder logic could generate a translation of RISC instructions in an invalid format. Unfortunately, the underlying RISC ISA for the decoder implemented for this study is regular, precluding such a scheme from being effective.

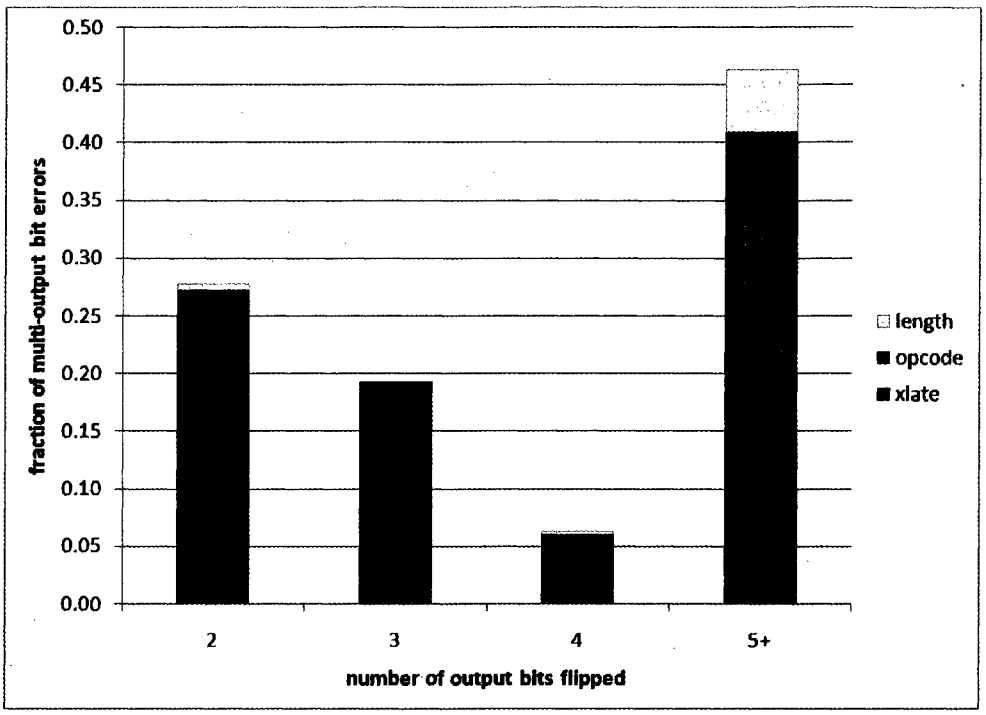


Figure 5.8: Characterization of Multi-bit Output Errors for z80 Decoder.

In summary, the results of the transient fault characterization performed on the z80 instruction decoder has shown that the assumptions made by many higher level reliability studies with respect to transient fault artifacts are not always valid. In particular, it has been shown that not only are output bits not corrupted with equal probability, but also that in many cases, multiple output bits can be corrupted simultaneously.

5.4 Mapping Protection Techniques to Logic Blocks

Finding the most appropriate technique (or class of techniques) to use to protect a particular unit is greatly dependent on the functionality and gate level structure of the particular logic block in question. In the context of logic, macro-level techniques generally provide protection implicitly, as any logic block within a microprocessor are often protected via spatial or temporal redundancy. While these techniques can generally be applied to all types of logic blocks, their associated overheads preclude their further evaluation in this dissertation work.

Property-based checking techniques are a natural fit for logic blocks in the datapath category, due to their narrow functionality and regular gate-level structure. Arithmetic circuits generally have regular structures, and compute simple operations whose correctness can easily be checked offline by verifying mathematical properties. Prior works by [72][28], where the correctness of integer arithmetic computation are checked through the use of residue codes are examples of property checking techniques being a good match with datapath logic blocks.

In contrast to property-based checking techniques, implementation level techniques are naturally suited to be applied to hybrid and control logic blocks. Logic blocks in this in these categories typically have more complex input to output mappings, whose correctness can not be as easily verified as datapath blocks. As an example, consider an integer multiplier (a datapath block) and the z80 decoder studied in this chapter (a control block). For the multiplier case, results can be

checked by performing modulo arithmetic on the multiplicands (essentially performing a much smaller multiplication operation), as discussed in [28]. For the z80 decoder case, it is significantly harder to determine if a sequence of RISC operations produced at the output of the logic block matches the z80 instruction provided at the input. In addition to this, control logic is likely to be significantly more random in structure than datapath logic, implying that some gates within such a logic block are more likely to logically mask faults. This is an attractive attribute because it implies that an implementation level technique can be effective while only manipulating a small subset of components. Considering the decoder again, there exists logic within that circuit which provides translations for instructions that rarely occur, which means transient faults originating at those nodes are likely to be masked. In contrast, a multiplier circuit, specifically one which sums partial products to produce a final result, should have very little logical masking occurring, as a transient fault affecting any partial result has a high probability of propagating.

5.5 Summary

In summary, this chapter has examined the task of identifying appropriate protection strategies for different types of logic blocks. Taxonomies for both classes of soft error tolerance solutions and different types of logic blocks were developed, allowing this task to be approached in a systematic manner. In addition to classifying logic blocks, the soft error vulnerability of a parallel instruction decoder

was characterized in detail. This characterization serves not only to justify the conclusion that implementation-level techniques are the a good fit for protecting control blocks, but also to illustrate that many assumptions made by prior works regarding the artifacts produced by transient faults are not valid in all cases.

Chapter 6

A Quantitative and Qualitative Approach to Protection and Analysis

6.1 Introduction

With the ultimate goal in mind of developing more effective ways to protect hybrid and control logic blocks, the final part of this study presents a new framework for transient characterization and analysis. This framework is proposed and evaluated in the context of the implementation-level class of mitigation techniques. Given the fact that implementation-level techniques achieve higher reliability by manipulating components on the gate or transistor level, there are typically numerous choices regarding which components should be protected first. In order to handle this, mitigation techniques in this category are typically coupled with heuristics whose purpose is to make qualitative decisions regarding which

elements should be protected first. The goal of such a heuristic is generally to provide some amount of benefit (higher reliability in this case), while minimizing some other important metric (area, cost, power, etc...). Typically, when qualitative methodologies like the one just described are employed, initial analysis is done to first select the components to be protected (in some ranked order), then additional analysis is performed to obtain a "cost-benefit" curve, indicating how much benefit could be had for a particular cost[11][79].

The work proposed in this section of the study expands the space of implementation level techniques by presenting a methodology that is both qualitative as well as quantitative. In addition to making qualitative decisions about which elements should be protected, our heuristic allows for immediate quantitative decisions to also be made regarding how much benefit could be had by protecting a particular element. The benefit of such a methodology is that it allows for component selection and a "cost-benefit" curve to be obtained in a single simulation pass, reducing the amount of effort required for analysis. Additionally, this technique is ideally suited for identifying particularly sensitive components within hybrid and control logic blocks, which can have complex fault propagation behavior.

6.2 Choosing an Implementation Level Technique

As was stated previously, implementation level techniques manipulate either individual transistors, combinational logic gates, or flip-flops to improve the reliability of logic blocks. In this work, our framework is proposed and evaluated in the con-

text of manipulating flip-flops. Flip-flops were chosen as the point of protection for two reasons. First, in most logic blocks, flip-flops are significantly smaller in number as well as total area consumed when compared to combinational logic gates. Also, detecting SETs at flip-flop inputs is attractive, as they are driven by a fan-in cone of combinational logic gates, meaning that the addition of detection logic at a single flip-flop can detect transients originating from many possible gates.

6.3 SET Detection and Correction

This section provides an overview of the SET detection techniques that could be selectively based on the results of the presented heuristic. Solutions are primarily considered which detect the presence of transients at flip-flop inputs by creating a duplicate copy of the master latch and comparing the values captured by both copies. Transients are actually detected by forcing the duplicate latch to capture its value at a slightly different time than the original. This strategy works because SETs manifest themselves in a manner similar to timing violations. The capture time of the duplicated master latch can be modified by either adding some additional delay on the data input path, or by using an altered clock waveform to control the latch. Examples of modified flip-flops using both strategies are shown in Figure 6.1. In each case, only the master latch in each flip-flop is duplicated, and the slave latch (not shown in either picture) does not need to be modified. Both solutions are conceptually similar, but each has unique advantages and dis-

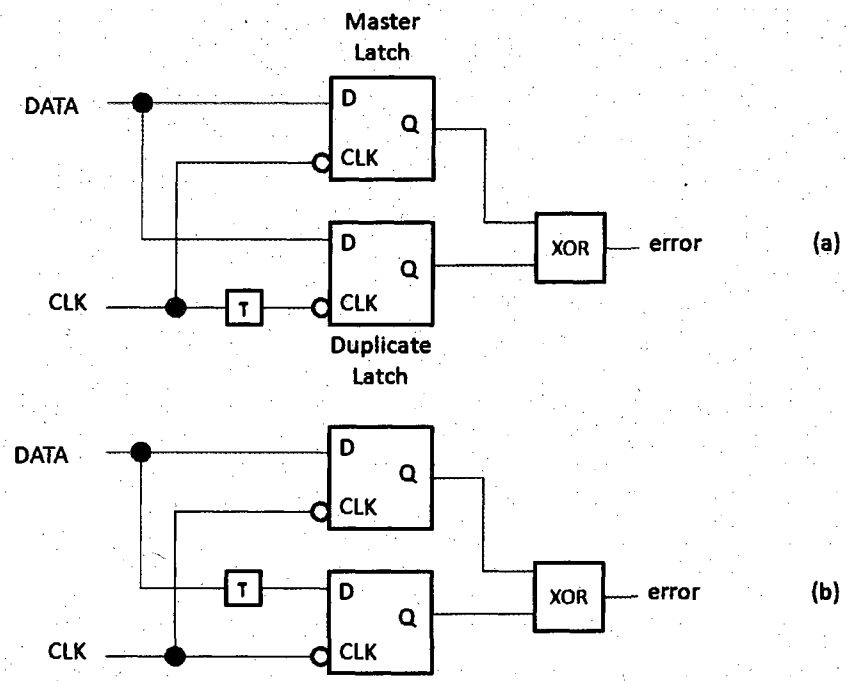


Figure 6.1: SET Detection via Master Latch Duplication.

advantages; these will be discussed in detail.

Time Shifted Clock Inputs

This method of SET detection was inspired by the Razor flip-flop proposed by Ernst et al. [14]. It should be noted that the original purpose of this flip-flop to allow for aggressive dynamic voltage scaling by detecting when timing violations occur. A timing diagram of how this technique detects the presence of errors is shown in Figure 6.2. The waveforms labeled C1 and C2 represent the clock inputs for the normal and redundant latches, respectively. Both latches take their data samples during the intervals specified by the vertical dotted lines (this work

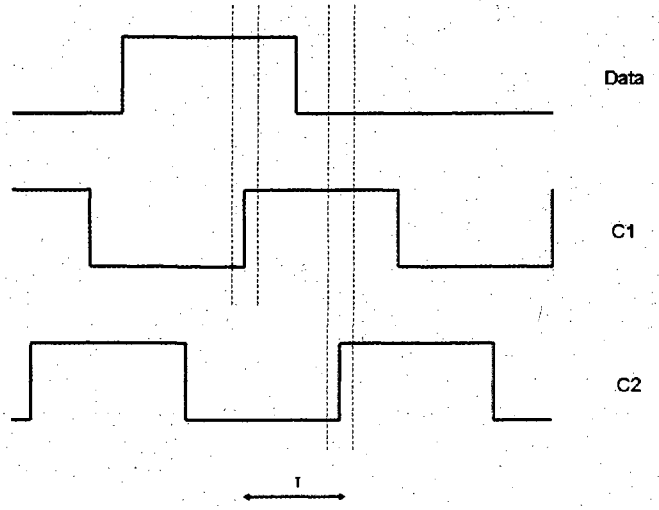


Figure 6.2: Timing Diagram for Time Shifted Clock SET Detection.

assumes positive edge triggered flip-flops). If there is a mismatch between the two data samples, the presence of an SET is detected and appropriate action can be taken. The main trade-off that must be considered when using a technique like this is related to how much delay is placed between the main and duplicate clock signals. A large amount of skew between the main and shadow clocks detects a greater fraction of propagating SETs, but can potentially create short path issues. If the skew between clocks is longer than the shortest path in the circuit, the data sample taken by the shadow flip-flop could be next unit of data propagating through the pipeline, potentially resulting in performance degrading false positives. The original Razor work dealt with this problem by manually padding short paths [14].

Time Shifted Data Inputs

Another equivalent method of SET detection is to time shift the data rather than the clock inputs to each latch. This method was inspired by the BISER work by Mitra et. al. [31]. In this case, each (the original and redundant latch) data input is being driven by the exact same fan-in cone of logic, except for an additional amount of delay placed on the input path of the redundant data input. This additional inserted delay results in each latch seeing shifted values during the rising edge of the clock (which is how SETs are detected). The timing diagram shown in Figure 6.3 illustrates how this technique can be used to detect errors. Like the previously presented solution, there also exists a trade-off concerning how much additional delay should be inserted between the original and redundant flip-flop. A large amount of delay can detect a greater fraction of SETs, but if the augmented flip-flop is on a critical path, the clock period must be increased.

It should be noted that while the work proposed in this chapter selectively augments flip-flops with BISER detectors, the original proposal intended to realize detectors through the modification of already present scan hardware [31]. Despite this original intention, the work presented in this chapter is useful because all microprocessors are not full scan, meaning that many sequential elements do not have corresponding scan elements. This is especially true for pipelined logic units, where scan hardware is unnecessary for internal ranks of flip-flops.

The best technique for a particular logic unit can vary depending on the characteristics of its timing paths. Time shifting the clock inputs is not an optimal solution for a circuit with a large number of short or zero delay paths as a sig-

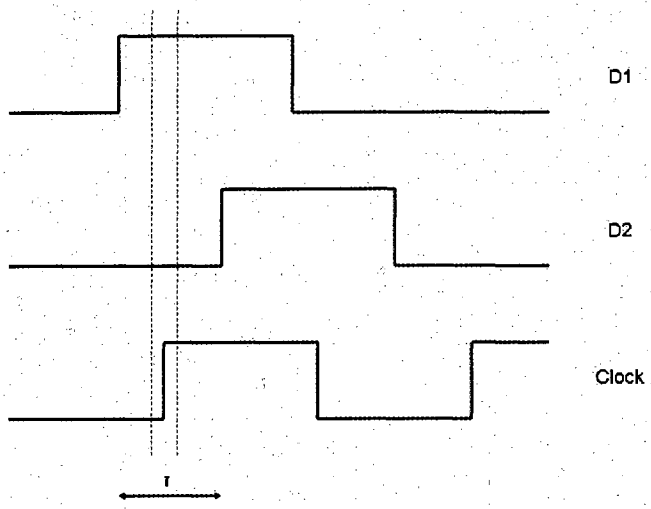
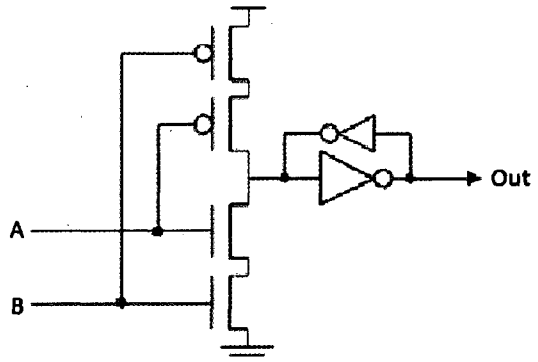


Figure 6.3: Timing Diagram for Time Shifted Data SET Detection.

nificant amount of delay padding would be required. In contrast, a circuit with balanced paths could potentially suffer a great deal of delay overhead (in terms of the minimum clock period achievable) if the data inputs were time shifted. For the purposes of this work, all detectors applied used time shifted data inputs, but we believe our results would be applicable for either approach.

Recovery

Another important issue that comes up during the design of a mitigation scheme is the action that should be taken after an error is detected. In general, a system can employ either backward or forward recovery upon detection of an error. When backward recovery is used, all computation occurring after the point where the error was detected is thrown away and redone. This typically requires some amount of check pointing and is best suited for macro-level mitigation techniques,



A	B	Out
0	0	0
0	1	previous state
1	0	previous state
1	1	1

Figure 6.4: Muller C-element with Keeper Circuit.

where errors originating from a variety of sources can be detected and recovered from in a uniform fashion. Because implementation level techniques are targeted toward protecting only a small subset of a system from faults, forward error recovery, where errors are corrected in place is an attractive option. In this work, both backward error recovery (where errors only need to be detected), and forward error recovery (where errors are detected and corrected) will both be explored. The detectors shown in 6.1 are sufficient for detection only and need to be modified in order to correct errors. The rest of this section will discuss these necessary modifications.

The BISER detectors, proposed by Mitra et. al. [31], correct errors in-place through the use of Muller C-elements. A C-element is a logic gate typically used within asynchronous circuits for synchronization [36]. A transistor level diagram of this logic gate (along with a keeper circuit) is shown in Figure 6.4.

C-elements act as inverters (buffers when the keeper circuit is considered)

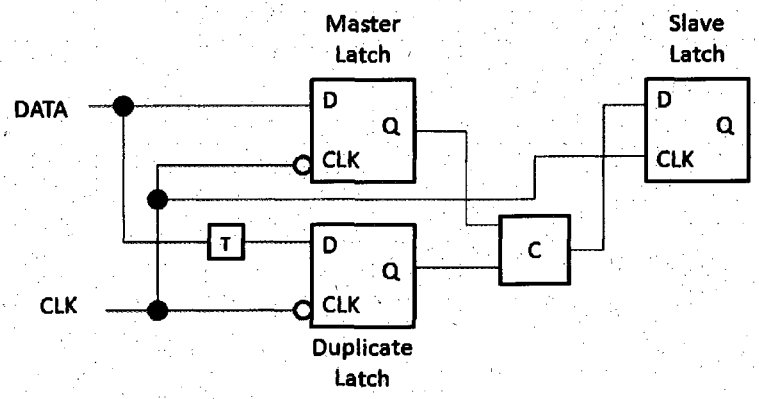


Figure 6.5: Error Correcting Flip-flop.

only when both inputs are identical. When the inputs are different values, the output of a C-element retains the output value determined by its previous input. Such a gate can be used to correct SETs arriving at flip-flop inputs in the following manner: Each data input (the original line going to the flip-flop, along with the delayed input going to the redundant copy for the flip-flop) is connected to the inputs of the C-element. Assuming that there is enough delay between the flip-flop inputs such that both inputs never glitch simultaneously, the output at the keeper will always be correct.

An error correcting flip-flop design is shown in Figure 6.5. In this design, the output of the original and duplicated master latches are fed into the C-element. The output of the C-element is then used to drive the slave latch.

The plot shown in Figure 6.6 illustrates how this correction can occur. In the figure three waveforms are shown. The top two wave forms represent the inputs to the C-element (which are also inputs to a normal and redundant flip-flop), while

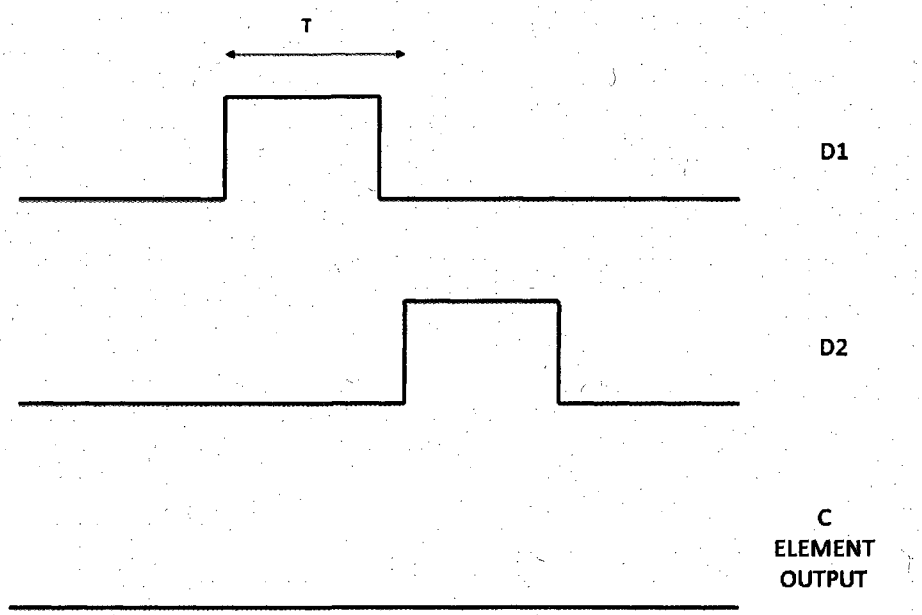


Figure 6.6: C-element Timing Diagram.

the bottom wave form shows the output of the keeper. In the scenario shown, the correct value is 0, and a striking particle as forced a 0-1-0 SET to propagate through the circuit. Because there is sufficient skew between the inputs, neither input to the C-element glitches at the same time, meaning that the output of the keeper (the bottom waveform) never changes. There is also some probability that the transient is long enough such that both C-element inputs glitch at the same time. In this case, both flip-flops will sample the wrong value, and the output of the keeper will also be wrong. When this happens, the corrupted state will end up propagating through the rest of the circuit.

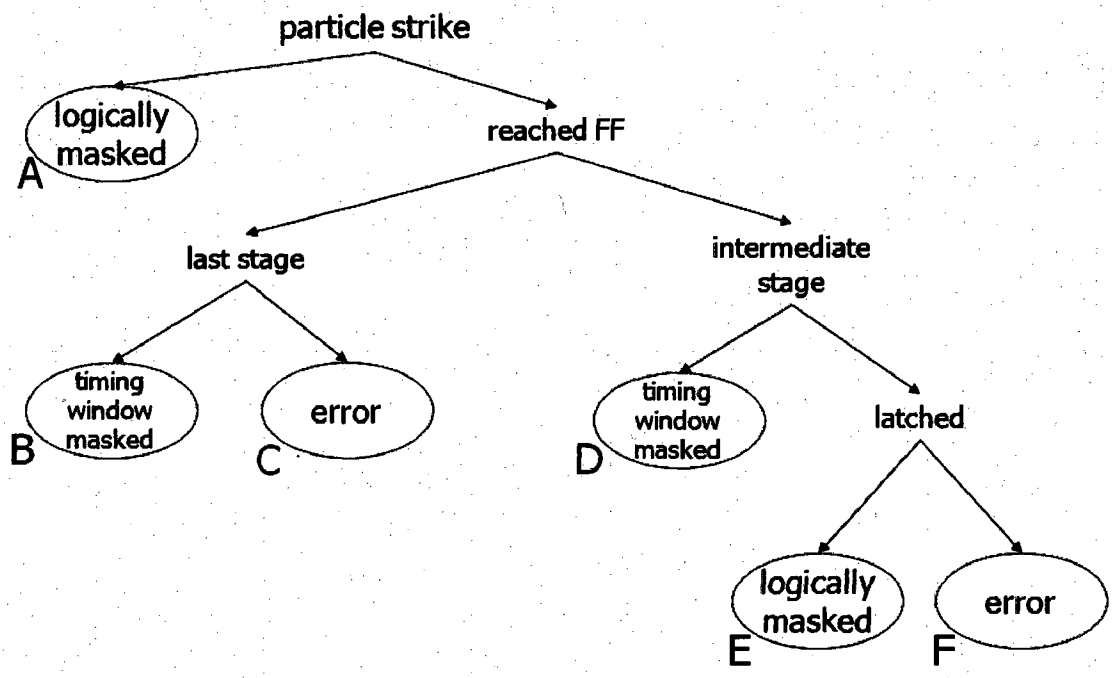


Figure 6.7: Logic Fault Outcome Tree.

Heuristic Motivation

As discussed previously, SETs only cause bit flips when they propagate from a combinational logic gate to an output and alter the value that is captured by a downstream flip-flop. As particle strikes occur with equal probability at any given point in time, individual output bits (flip-flops) in a circuit timing window mask SETs uniformly. In contrast, individual output bits can have differing fan-in cones, meaning that SETs can potentially propagate to individual output bits at varying rates. This essentially means that in contrast to timing window

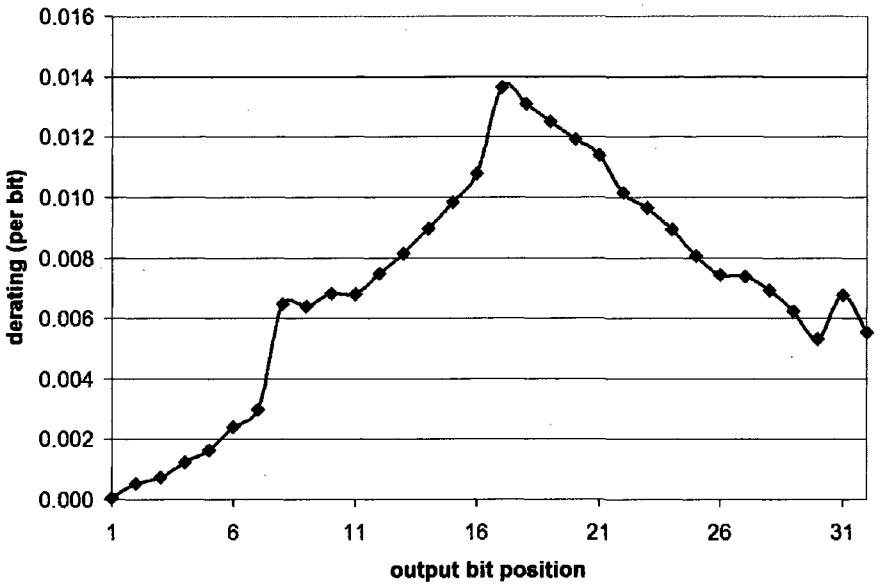


Figure 6.8: 16x16 Multiplier Derating per Bit.

masking, logical masking is not necessarily uniform across output bits. A prior study on estimating SER reports that in multipliers the center bits tend to have an error rate that is orders of magnitude larger than those of the bits closer to the most and least significant positions [76]. The authors of this work refer to this phenomenon as SER peaking [76]. We have also observed this phenomenon by modeling a combinational 16x16 integer multiplier and performing statistical fault injection. Figure 6.8 shows the amount of errors that occur on each output bit of the multiplier. We believe that this SER peaking phenomena presents an opportunity for low cost soft error protection. Ideally, a combinational multiplier with this behavior could cost-effectively be hardened from logic soft errors by simply protecting the subset of output flip-flops where SER peaking occurs.

For combinational circuits, the subset of output flip-flops that need to be protected can be identified by performing statistical fault injection and observing the number of times each output bit is corrupted. Identifying a similar subset of flip-flops in a pipelined circuit is a significantly harder problem. Figure 6.7 shows our assumed fault model for a SET occurring in a pipelined circuit. This is significantly more complex than the model for a SET in a combinational circuit, which would only consist of outcomes A, B, and C. From this model, it is clear that even if a SET propagates to and is captured by a flip-flop, that error could still potentially be logically masked as it propagates through the ensuing pipeline stages, never manifesting itself at a circuit output. In addition to this, it is also possible for an SET to corrupt multiple intermediate flip-flops in a circuit, and have only a subset of the corrupted elements be responsible for propagating that error to the outputs. Examples of such scenarios will be provided during the presentation of the proposed heuristic. The methodology presented in this work accurately identifies the flip-flops, in intermediate ranks as well as outputs, which most significantly impact the failure rate (and thus are the best location to place SET detectors) in the context of this more complex fault model.

Flip-Flop Selection

In this section, the heuristic for selecting flip-flops is presented. Prior to statistical fault injection, each flip-flop in the circuit is allocated a counter. This counter represents the overall contribution (of the corresponding flip-flop) to the circuit failure rate. The pseudo-code for our proposed selection heuristic is shown in

```
1 for (each fault injected)
2  if (error)
3   if (case C)  /* strike in last stage of logic */
4    compute set P /* set of all outputs flipped */
5    score_inc = 1 / cardinality(P)
6    increment counter for each member of P by score_inc
7   else if (case F) /* strike in an intermediate stage */
8    compute set S /* set of flipflops which propagated error */
9    score_inc = 1 / cardinality(S)
10   increment counter for each member of S by score_inc
11
12 sort counters in descending order
```

Figure 6.9: Selection Heuristic Pseudo-code.

Figure 6.9. Referencing the fault model shown in Figure 6.7, an error is defined as a particle strike which results in either outcome C (a SET occurring in the last stage of logic and subsequently altering the value of captured by an output flip-flop) or outcome F (a SET occurring in an intermediate stage and propagating to an output flip-flop).

An example of outcome F (represented by lines 7-10 in Figure 6.9) is shown in Figure 6.10. In this example, a SET occurring in the first pipeline state ends up propagating and corrupting the values captured by flip-flops 1, 2, and 3. In the next clock cycle, the erroneous values launched from flip-flops 1 and 2 end up propagating and corrupting output bits 4 and 6. The value launched from flip-flop 5, on the other hand, is logically masked. In this case, the set P (defined as all flip-flops that stored incorrect values), contains flip-flops 1, 2, 3, 4, and 6. Set S (referenced in line 8 in Figure 6.9) represents flip-flops that in addition to capturing a transient value, are responsible for propagating incorrect values to

circuit outputs. For this injected fault, only flip-flops 1 and 2 belong to set S, and protecting both guarantees that this fault will not propagate. Set S is computed by back propagating along the the D-frontier (definition and ref) from all corrupted outputs. The counter for each flip-flop belonging to set S is then incremented appropriately.

From the statement shown in line 9 of Figure 6.9 it is clear that amount a flip-flops counter get incremented for propagating an error is directly dependent on the size of set S. In cases where a smaller number of flip-flops are responsible for propagating a fault, the members of S will be incremented by a larger value. The reason for this is because a primary goal of this methodology was to try and minimize area, and preventing faults which are propagated by a single (or small number) of bits flips is the most cost effective approach in this regard.

At the end of the characterization run each counter contains (for its corresponding flip-flop) the overall contribution in terms of the total number of errors observed during fault injection. The value stored by each counter represents an approximation of the number of times a flip-flop is responsible for either directly causing an error by capturing a transient value (output flip-flops in the last stage) or indirectly causing an error by capturing a transient value and logically propagating that value to a circuit output (flip-flops in intermediate stages). A counter with a high value implies that the associated flip-flop is more likely to capture and/or logically propagate a transient value, and thus would be an ideal candidate for protection. Sorting these counters (performed on line 12 of Figure 6.9) creates a list of flip-flops ranked according how much of an overall benefit could be

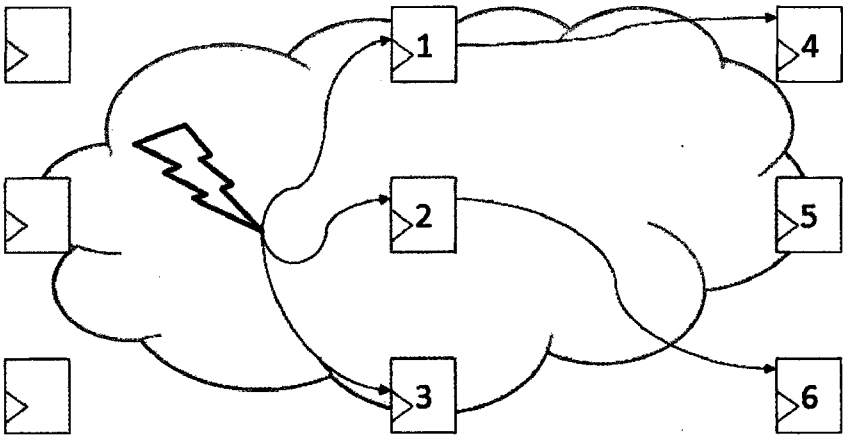


Figure 6.10: Example of SET in an Intermediate Pipeline Stage.

obtained by augmenting a particular sequential element with a soft error detector. This obtained ranking allows for quantitative decisions to be made regarding which flip-flops should be protected first.

The ranking technique presented in this work is unique in that in addition to identifying which flip-flops are the most likely to capture and propagate transient values, this technique also gives a quantitatively accurate estimate of how much protecting each flip-flop impacts the overall error rate. Quantitative accuracy is achieved through the counting policy employed by the presented heuristic. As

stated previously, the value held by each counter represents the total number of errors observed that were caused either directly or indirectly by the corresponding flip-flop. The sum of all of the counters represents the total number of errors that were observed during the fault injection. Dividing the counter value of a flip-flop (or the sum of values of a group of flip-flops) by the total number of errors observed yields a percentage which represents a prediction of error coverage, or what fraction of observed errors could be eliminated if the flip-flop (or group of flip-flops) was augmented with SET detection and correction logic.

6.4 Results

In this section the proposed quantitative and qualitative methodology is evaluated. Three benchmarks, the parallel z80 decoder, a double precision floating point adder, and a pipelined integer multiplier were chosen for evaluation. These benchmarks were chosen because together they represent all classes of the previously defined logic block taxonomy. The decoder, adder, and multiplier each represent the control, hybrid, and datapath logic block classes, respectively.

Predicted Error Coverage

Predicted error coverage estimates yielded by the presented heuristic are shown in Figure 6.11. The x-axis represents for each circuit the fraction of flip-flops augmented with detection logic, and the y-axis represents the predicted amount of error coverage that can be gained by protecting that fraction of flip-flops. Each

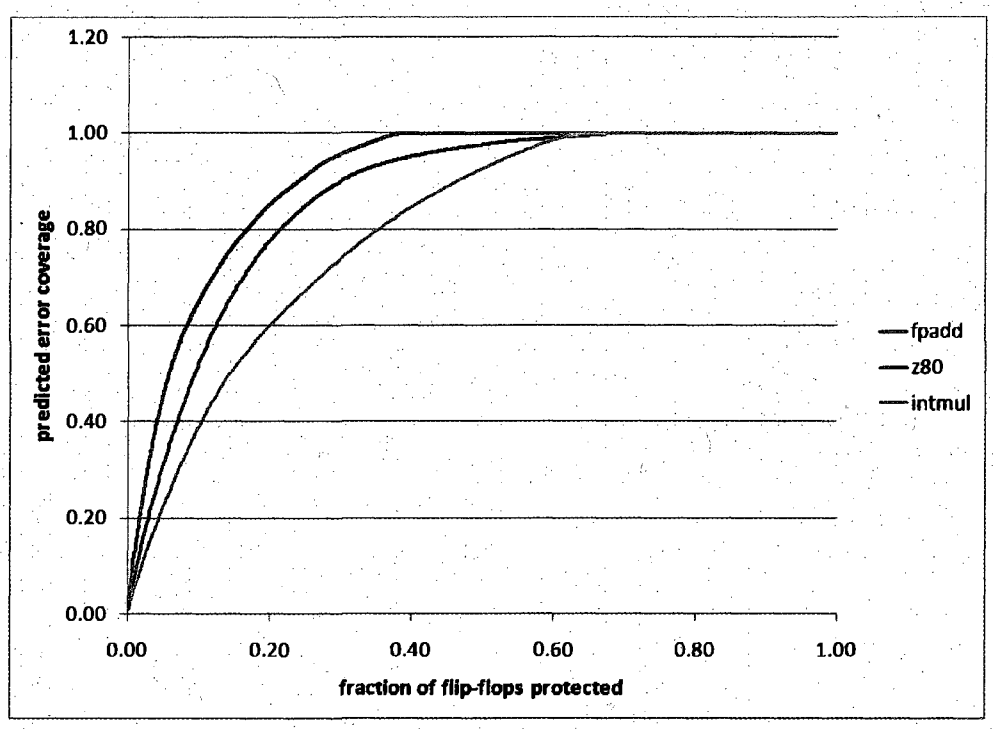


Figure 6.11: Predicted Error Coverage.

individual point in the curves shown is obtained by first taking the summation of counter values corresponding to the fraction of flip-flops being protected, and then dividing the by the total number of errors observed during fault injection for that benchmark. The fraction of sequential elements being protected corresponds to the x-coordinate of each point, while the previously defined ration represents the y-coordinate.

Looking at all of the curves shown in Figure 6.11, it is clear that all flip-flops in each circuit are not equal in terms of the amount of error coverage that can potentially be obtained through the placement of an SET detector. If all flip-flops within each circuit were indeed equivalent in this regard, each of the three curves

shown in Figure 6.11 would be straight lines. The integer multiplier, represented by the dotted green line in the figure, is flatter than the other curves. The reason for this can be attributed to the highly regular structure of this particular circuit. This implementation is fairly straightforward, with partial products first being generated and then added together to obtain the final product, meaning SETs can propagate to many flip-flops in this circuit with a high probability.

For the z80 decoder, the most vulnerable state bits are primarily the output flip-flops in the last stage of the pipeline. While the z80 decoder implemented for this study is divided into length decode, opcode identification, and translation stages, the translation stage is the largest in terms of area. Additionally, the flip-flops storing decoded opcodes following the second pipeline stage are also responsible for propagating a significant number of transient faults.

In the floating point adder flip-flops present in the intermediate stages of the the adder, responsible for holding the aligned fractions, at the end of stage 2, and the sum of the added fractions (following stage 3) have the most potential benefit in terms of the overall number of errors observed if augmented with SET detectors. Despite the fact that the logic block taxonomy proposed for this study defined the floating point adder as a hybrid block, the datapath for this circuit is significantly wider than the control path, meaning that SETs are more likely to propagate to datapath flip-flops.

Verification

In addition to presenting results on predicted error coverage yielded by the proposed heuristic, a second set of experiments was performed in order to verify the accuracy of this coverage estimate. The purpose of these experiments was to essentially validate the initial claim that the heuristic presented is quantitatively accurate. It is important to note that this verification step is not required for the proposed heuristic to be used, but a step like this would be required to obtain a "cost-benefit" curve if any previously proposed qualitative methodology was employed. In order to perform this verification step, the quantitative information given by our heuristic, the rankings of flip-flops in order of the most important to protect, is taken and divided up into subsets: a baseline set (containing no flip-flops), the top 5%, 10%, 25%, 50%, and 100% of flip-flops. Another set of fault injections is performed (using a different random seed from the initial characterization experiment), augmenting each flip-flop in the previously defined subsets with a SET detector. For these experiments, real error coverage is plotted, which is defined as the fraction of errors that occur in the baseline set case (where nothing is protected), that would have been protected by the SET detectors in one of the other subset cases. Also the detectors assumed for this verification are ideal, meaning that a sufficient amount of delay is placed between the data inputs of the normal and redundant flip-flops used in order to make sure that all arriving transients can be detected and corrected. Adding this additional delay, means that in some cases the clock period for the unit being protected may have to be increased. The trade-off between clock cycle time, and the amount of protection

that can be obtained is further explored later in a separate section. Both forward and backward error recovery are considered in the verification experiments conducted in this section.

Figures 6.12, 6.13, and 6.14 plot real vs predicted error coverage for the z80 decoder, floating point adder, and integer multiplier, respectively. In each figure, 3 curves are plotted representing the predicted error coverage (shown previously in Figure 6.11), the measured real coverage assuming backward error recovery, and the measured real coverage assuming forward error recovery. For the experiments assuming backward error recovery, as long as at least one flip-flop capturing an incorrect value during the injection of a transient fault is protected, the error (assuming the transient end up propagating to a primary output) is counted as prevented. In the forward error recovery case, an error is not counted as prevented unless all flip-flops in the critical set responsible for propagating the error are protected.

The results of this verification experiment on the z80 decoder are shown in Figure 6.12. In this figure the dotted line represents the predicted error coverage (previously plotted in Figure 6.11), while the two solid lines represent the results of the validations experiments. For this particular benchmark circuit, the predicted error coverage very closely track the measured real coverage.

For the floating point adder, both predicted and real error coverage are plotted together in Figure 6.13. For this benchmark, the predicted and and real error coverage are relatively closely correlated, but not as closely as what was observed for the decoder circuit.

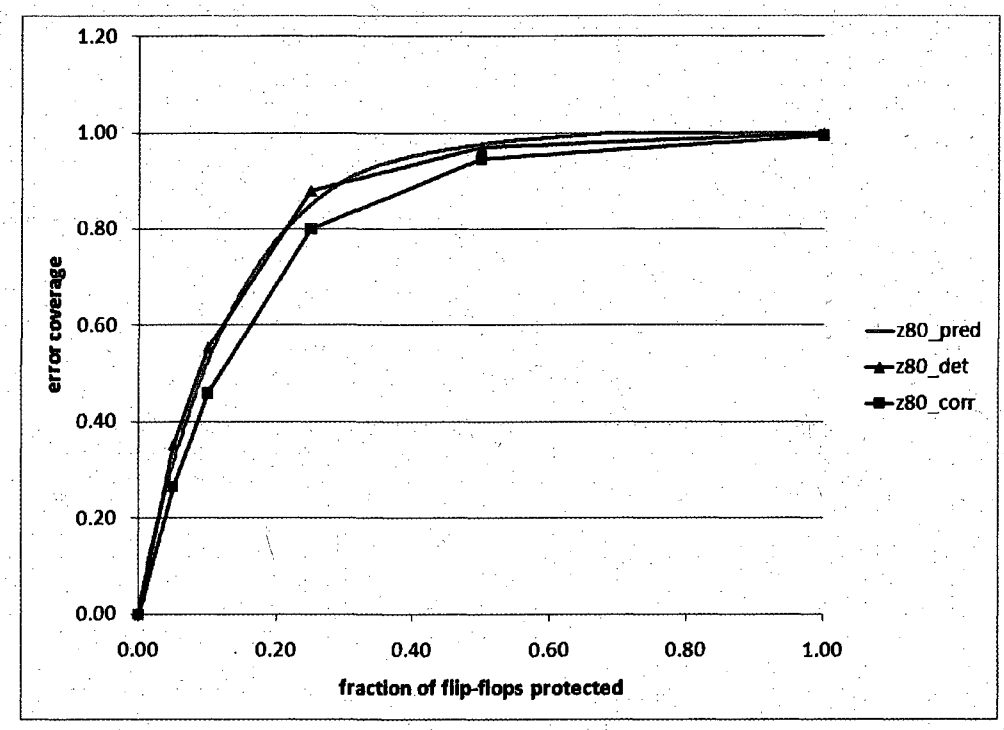


Figure 6.12: Predicted vs. Real Error Coverage for z80 Decoder.

Figure 6.14 shows the results of the verification experiments conducted on the integer multiplier. For this circuit, it is clear that the real coverage plotted does not correlate as closely to the predicted error coverage as observed for the decoder and floating point adder, particular for the correction case. For this case, the correction curve is actually slightly convex in the middle of the plot, implying that in this range our heuristic is making the wrong decisions about which flip-flops to protect. This can be attributed to the structural regularity of the integer multiplier, meaning that many of the flip-flops in this circuit have similar counter values.

For both the hybrid and control logic blocks used in the evaluation of the

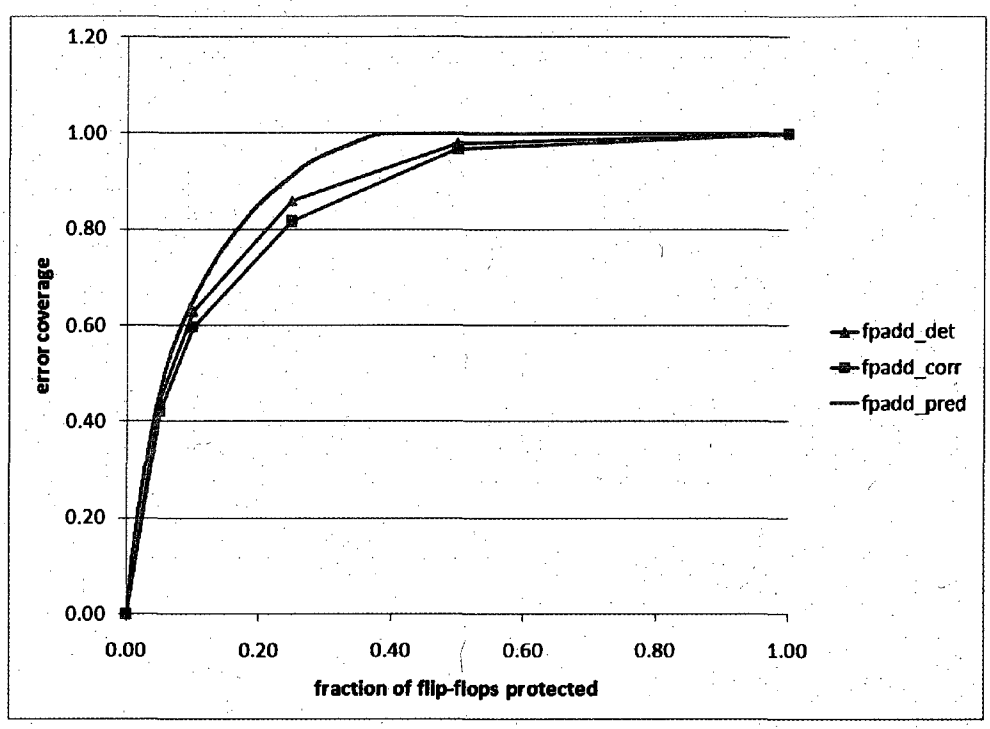


Figure 6.13: Predicted vs. Real Error Coverage for the Floating Point Adder.

presented methodology, the real coverage measurements obtained from the verification experiments correlate closely to the predicted error coverage obtained from the selection heuristic. This means that a designer using this methodology to analyze (and ultimately protect) a logic block can rely exclusively on the predicted error coverage given by the presented methodology (which only requires one experiment), rather than also having to perform this additional verification step (which requires several more experiments).

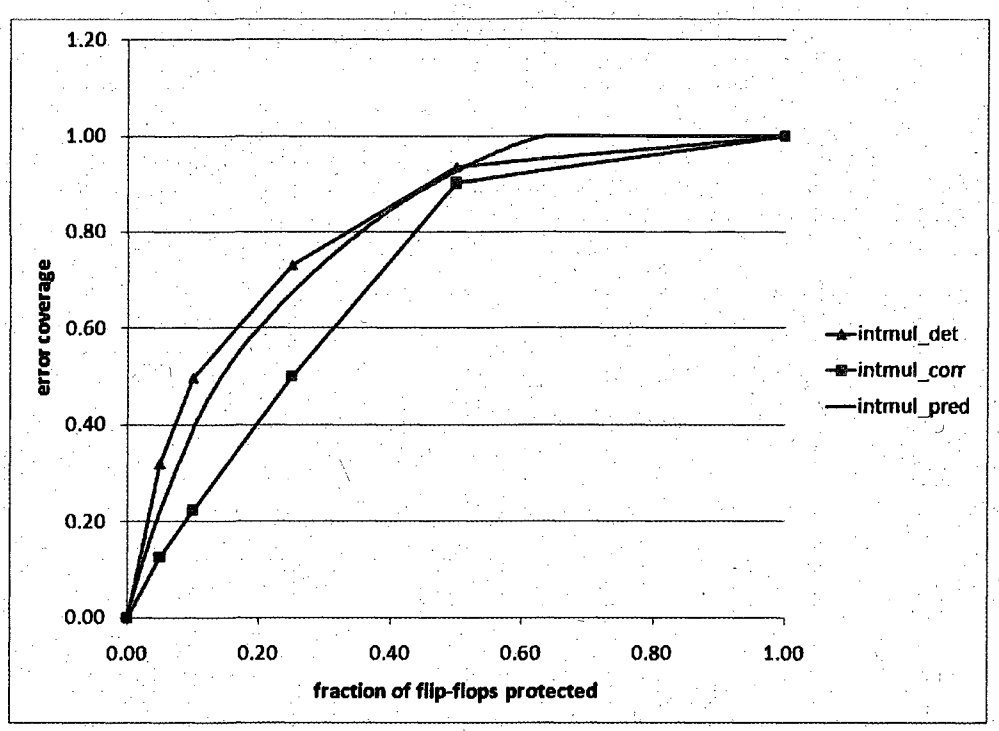


Figure 6.14: Predicted vs. Real Error Coverage for Integer Multiplier.

Approximation Bias

While the measured real coverage does correlate closely with the predicted error coverage, some approximation bias does exist. Looking at the graphs in the previous section, the predicted error coverage always overestimates the measured real error coverage for the correction case. In addition to this, the measured real error coverage for the detection case is usually larger than the coverage measured when forward error recovery is assumed.

This bias can be attributed to a small subset of cases similar to the situation shown in Figure 6.10. Recall that this figure depicts a transient fault occurring in

an intermediate pipeline stage, with multiple flip-flops following that intermediate stage being responsible for propagating erroneous values through the rest of the circuit. If only a subset of those previously mentioned flip-flops responsible for propagating the error are protected, a discrepancy in the calculated error coverage between the heuristic and validation experiment will occur. For the purposes of this discussion, assume that only candidate flip-flop 1 is protected.

The heuristic presented in this work treats each flip-flop in a circuit as an individual entity. When the predicted error coverage is calculated (corresponding to having a subset of flip-flops protected), the sum of all the counters corresponding to flip-flops in the subset is divided by the total number of errors observed. In this particular case, the predicted error coverage would be (1/2), as the counter for flip-flop 1 has a value of (1/2), and 1 total error was observed. Assuming only error detection (backwards error recovery), if an identical fault (on the same gate with the same input stimulus) occurred during the verification run, the measured real coverage would be 1, because a transient fault was detected at the input of at least of on the flip-flops in the circuit (in this case flip-flop 1). In contrast, for the correction case, the measured real coverage would be 0, because flip-flop 2 was unprotected, allowing erroneous values to still propagate to primary outputs.

Overhead

As was stated previously, the fraction of transients that can be detected with detectors like the one shown in Figure 6.1 depends primarily on the amount of additional delay inserted between the master and redundant latch. The drawing

in Figure 6.15 illustrates the relationship between transient width (denoted as duration in this diagram), and the inserted delay between detectors (denoted as delay). In this diagram, two waveforms are shown, representing the inputs to the master and redundant latch in the proposed SET detector, respectively. In both waveforms a 0-1-0 transient is present, with the transient being skewed by delay time units for the input to the redundant latch. The timing diagram shown is also divided in into three regions based on what would happen if the rising edge of the clock occurred during that particular period of time. Region I represents the situation where both inputs were correct initially, and then only the input to the master latch glitched due to the presence of a SET. In this case, the transient can not only be detected, but can also be corrected, because the C-element (if we are assuming correction) would not have changed it value when the master input glitched. In Region II, both the master and redundant latch inputs have glitched to the wrong value. In this situation, a transient will never be detected because both the master and redundant latch inputs agree, meaning the output of the XOR gate in Figure 6.1 will stay at zero. Region III represents the case were both inputs have glitched, but the input to the master latch has returned back to the correct value. In this case, it is possible to detect an error (since both inputs do not match), but impossible to correct an error (because the value stored by the C-element will be wrong). If detection alone is assumed (implying backward error recovery), the rising clock edge occurring in Region III represents a false positive, because in the unprotected case, the transient fault would have been timing window masked. If forward error recovery is assumed, this same situation could be problematic,

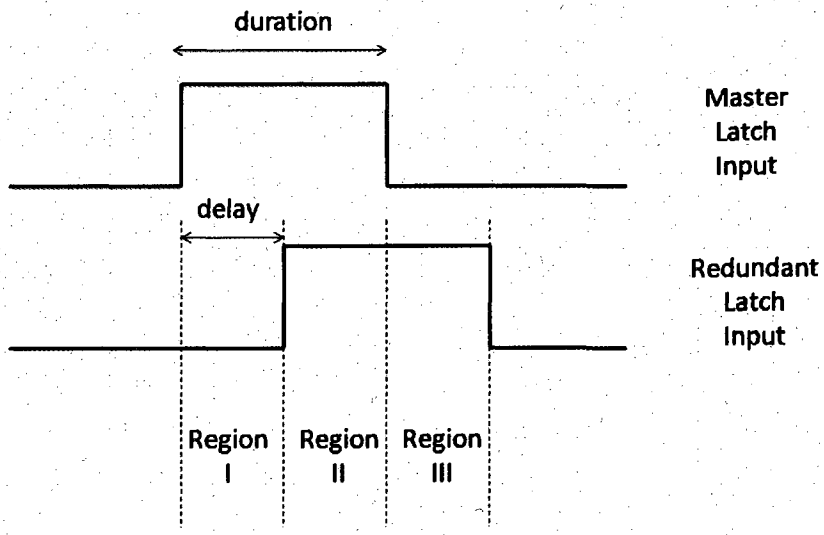


Figure 6.15: Relationship Between Inserted Delay and Probability of Transient Detection.

as a protected flip-flop would "correct" the output bit to the wrong value.

Assuming that transients arrive at flip-flop inputs at any given time with equal likelihood, the probability that a transient is detected or corrected is $\frac{delay}{duration}$. For the detection case, transients falling in Region III can be ignored, because these are false positives which only affect performance. For the correction case, the fraction of Region III transients that don't result in errors can be expressed by multiplying $\frac{delay}{duration+delay}$ by the probability of logical masking.

The previously described relationships between inserted delay and transient width can be used to approximate the effects of having detectors with less than ideal delay (meaning that some fraction of transients are not detected). The dis-

crete probability function used for injected transient width (described in Chapter 3) can be scaled in order to calculate the fraction of transients that that would be detected assuming a given amount of inserted delay. This fraction can be calculated using the expression given in Equation 6.1.

$$\text{scaled\_coverage} = \sum_{i=0}^{n} w_i * min(\frac{delay}{duration}, 1) \quad (6.1)$$

The predicted error coverage measurements presented earlier in this chapter represent the error coverage attainable if there was enough inserted delay to detect all arriving transients. The scaled_coverage value yielded by the expression shown in Equation 6.1 represents what fraction of that attainable coverage can be had by having a SET detector with a smaller amount of inserted delay.

Combining the outlined strategy for scaling error coverage with area estimates obtained through synthesis allows the simultaneous comparison of error coverage, delay, and area overhead. As was discussed in Chapter 3, all benchmarks circuits studied in this thesis were synthesized into LSI10k standard cell library gates. The reported area from this synthesis was used as the baseline area, and the cost of each detector was defined to be the area of a latch (representing the duplicated master latch in Figure 6.1) plus four inverters (representing the C-element and keeper logic). For the delay inserted between latches within a detector, the amount of delay is increased in increments of 20 picoseconds.

The three-dimensional plot shown in Figure 6.16 illustrates the trade-offs between error coverage, delay, and area overhead. The z-axis in this figure rep-

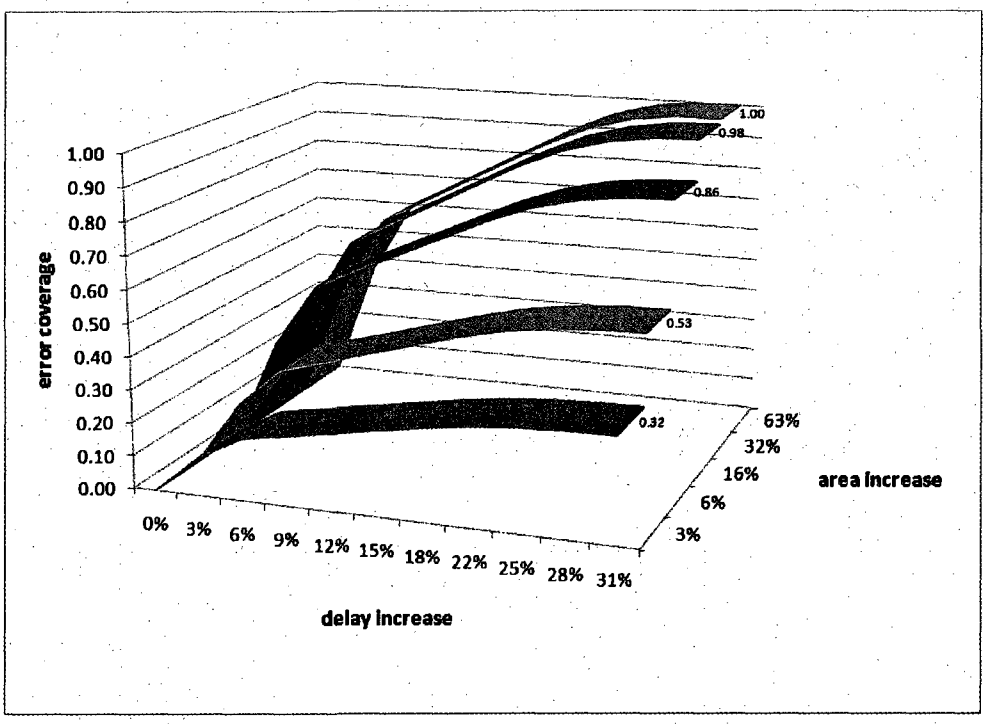


Figure 6.16: Tradeoff Between Area, Delay, and Error Coverage for z80 Decoder.

resents error coverage, while the x and y-axes represent delay and area overhead respectively. In addition to this, the data label displayed at the end of each curve represent the amount of error coverage achievable using the maximum delay penalty. The labeled set of points in this figure correspond to the reported predicted error coverage for the z80 decoder (shown in Figure 6.11). From this plot it is clear that a desired level of error coverage can be achieved via multiple combinations of area and delay overhead.

6.5 Summary

In this chapter, a framework for combinational logic soft error analysis was presented. This framework is unique in that it is quantitative as well as qualitative, allowing rapid "cost-benefit" trade-offs relating to reliability to be made. The framework is specifically targeted towards protecting hybrid and control logic blocks in a cost-effective manner, and achieving this goal through the modification of sequential elements. The results presented in this chapter illustrate the vast space of design possibilities in terms of delay, area, and reliability.

Chapter 7

Conclusion

The work completed in this dissertation is motivated by several factors. One of the major reasons soft errors in logic are becoming a more significant design concern is because of technology scaling. While critical charge values for transistors within both SRAM cells and logic gates are shrinking because of smaller feature sizes, Q_{crit} values for combinational logic gates are shrinking at a faster rate because of the additional effects of diminished electrical and timing window masking. Independent of technology trends, the issue of soft errors in logic is also garnering an increased amount of attention because protection techniques for storage are already used extensively in current generation chips. This pervasive use of storage protection means that an increasing fraction of the on-chip die area vulnerable to particle strikes belongs to transistors within logic blocks. In addition to this, macro-level redundancy schemes often have performance, power, area, and verification overheads that make their use prohibitive in many design

spaces.

These motivating factors inspired the work presented in this dissertation, which represents a successful attempt at truly understanding the effects of soft errors in logic, and what those effects imply on an architectural level. In the process of completing this study, several important insights related to logic SER were uncovered:

- Contrary to conventional intuition, the overall impact of transient faults on logic (combinational logic gates and sequential elements) is largely independent of pipeline depth. In addition to this, transient faults actually have a smaller impact on combinational logic within deeper pipelines. As the analysis in Chapter 4 shows, this flawed intuition stems from two sources: the use of rates as direct comparison point (which is not appropriate in this situation) and a second order effect relating to how SETs fan-out along multiple combinational paths. This second order effect has a significant effect on the rate in which logic blocks timing window mask faults, and is obscured unless timing window masking is modeled in detail.

- Several commonly held assumptions about the manner in which transient faults manifest themselves have been shown to not be valid in all cases. The results presented in chapters 4 and 5 show than particle strikes on combinational logic gates often do not result in a single state element (whether internally or at a primary output) being flipped. Additionally, the study done in chapter 5 shows that within a pipelined unit, not all state elements

are corrupted with the same probability, and those which are corrupted are logically masked at varying rates.

These insights help answer questions related to how to think about this problem, which effects are most important to model, and what the structure is of the artifacts left behind by these faults. In addition to these conceptual insights, an analysis framework was presented in Chapter 6 that is both quantitative and qualitative. This methodology is valuable in that it facilitates rapid "cost-benefit" analysis to be performed, and accounts for the complex manner in which many faults propagate.

The combination of the insights and other contributions made by this thesis should ultimately provide for architects (and other engineers working at layers of abstraction above the gate level) a groundwork for understanding the appropriate manner in which logic soft errors should be attacked.

7.1 Future Work

The work done in this dissertation can be expanded in several directions. While the results presented in this thesis have exclusively focused on the problem of soft errors, the tools developed can be extended to study other modes of failure. Problems such as wear-out, environmental and process variations, and manufacturing-related defects are also significant design concerns and warrant additional study.

In terms of developing new techniques for transient error tolerance, the presented work can be extended in a few directions. The analysis in Chapter 5

can be leveraged at a design level in order to decide how a logic block can be redesigned at a high level in order to provide error tolerance. The z80 decoder studied in Chapter 5 provides an example of this opportunity. One of the major conclusions of the characterization of that particular logic block was that because of the regularity of the underlying RISC ISA that instructions were translated to, error detection would be difficult. One possible improvement that could be made would be to intentionally design some irregularity into this ISA in order to allow for simpler detection. This irregularity could come in the form of a larger space of invalid op-code register combinations, specific bit encoding patterns for opcode and register identifiers, or a number of other options.

Perhaps the most significant extension of this work is how the presented insights could be used to improve higher level studies on fault tolerance. All of the insights regarding the effects of soft errors in logic came about because the problem was studied at the gate and circuit levels of abstraction. Despite this, there are still some situations (especially early in the design cycle or during software development), where higher level models (behavioral RTL, performance simulators, or even binary instrumentation tools) may need to be used to study the effects of faults. The lessons learned during the course of this dissertation can serve to improve the effectiveness of these tools by guiding the assumptions made with respect to modeling faults.

For example, in the context of application fault injection studies the effect of a soft error is commonly modeled as a single bit corrupted in the computed result (if a functional unit is affected), or a single bit corruption in one of the instruction

specifying fields (if the fault in in either storage element or the decoder). Instead of an SEU-based model, based on the work presented in this dissertation, the functional unit fault case could be modeled by identifying clusters of output bits which are likely to flip together, while decoder faults could be modeled as a transformation from the correct stream of micro-operations to a alternative incorrect stream.

Bibliography

[1] Design Panel for SELSE Workshop 2006.

[2] HSPICE PTM – http://www.eas.asu.edu/ ptm.

[3] D. A. Anderson and G. Metze. Design of Totally Self-Checking Check Circuits for m-Out-of-n Codes. IEEE Transactions on Computers, 22(3):263–269, 1973.

[4] Hisashige Ando, Yuuji Yoshida, Aiichiro Inoue, Itsumi Sugiyama, Takeo Asakawa, Kuniki Morita, Toshiyuki Muta, Tsuyoshi Motokurumada, Seishi Okada, Hideo Yamashita, Yoshihiko Satsukawa, Akihiko Konmoto, Ryouichi Yamashita, and Hiroyuki Sugiyama. A 1.3GHz Fifth Generation SPARC64 Microprocessor. In DAC '03: Proceedings of the 40th Conference on Design Automation, 2003.

[5] Ghazanfar Asadi and Mehdi B. Tahoori. An Accurate SER Estimation Method Based on Propagation Probability. In DATE '05: Proceedings of the conference on Design, Automation and Test in Europe, 2005.

[6] Todd M. Austin. DIVA: a Reliable Substrate for Deep Submicron Microarchitecture Design. In MICRO 32: Proceedings of the 32nd annual ACM/IEEE international symposium on Microarchitecture, 1999.

[7] Gordon B. Bell and Mikko H. Lipasti. Skewed Redundancy. In PACT '08: Proceedings of the 17th International Conference on Parallel Architecture and Compilation Techniques, October 2008.

[8] David Bernick, Bill Bruckert, Paul Del Vigna, David Garcia, Robert Jardine, Jim Klecka, and Jim Smullen. NonStop Advanced Architecture. In DSN '05: Proceedings of the 2005 International Conference on Dependable Systems and Networks, 2005.

[9] Michael Orshansky Bin Zhang, Wei-Shen Wang. FASER: Fast Analysis of Soft Error Susceptibility for Cell-Based Designs. In Proceedings of ISQED 2006, April 2006.

[10] Arijit Biswas, Paul Racunas, Razvan Cheveresan, Joel S. Emer, Shubhendu S. Mukherjee, and Ram Rangan. Computing Architectural Vulnerability Factors for Address-Based Structures. In Proceedings of ISCA-32, 2005.

[11] Jason Blome, Shantanu Gupta, Shuguang Feng, Scott Mahlke, and Daryl Bradley. Cost-Efficient Soft Error Protection for Embedded Microprocessors. In Proceedings of International Conference on Compilers Architecture Synthesis for Embedded Systems, October 2006.

[12] J.J. Cook and C. Zilles. A characterization of instruction-level error derating and its implications for error detection. In Proceedings of DSN 2008, June 2008.

[13] H. Deogun, D. Sylvester, and D. Blaauw. Gate-level Mitigation Techniques for Neutron-induced Soft Error Rate. In Proceedings of ISQED, March 2005.

[14] Daniel Ernst, Nam Sung Kim, Shidhartha Das, Sanjay Pant, Toan Pham, Rajeev Rao, Conrad Ziesler, David Blaauw, Todd Austin, and Trevor Mudge. Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation. In Proceedings of MICRO-36, November 2003.

[15] L. B. Freeman. Critical Charge Calculations for a Bipolar SRAM Array. IBM J. Res. Dev., 40(1):119–129, 1996.

[16] B. Greskamp and J. Torrellas. Paceline: Improving Single-Thread Performance in Nanoscale CMPs through Core Overclocking. In PACT '07: 16th International Conference on Parallel Architecture and Compilation Techniques, September 2007.

[17] M.C. Hansen, H. Yalcin, and J.P. Hayes. Unveiling the ISCAS-85 Benchmarks, a Case Study in Reverse Engineering. IEEE Design and Test, 16(3):72–80, 1999.

[18] John P. Hayes, Ilia Polian, and Bernd Becker. An Analysis Framework for Transient-Error Tolerance. In VTS '07: Proceedings of the 25th IEEE VLSI Test Symmposium, 2007.

[19] P. Hazucha and C. Svensson. Impact of CMOS Technology Scaling on the Atmospheric Neutron Soft Error Rate. IEEE Transactions on Nuclear Science, 47(6):2586–2594, December 2000.

[20] Robert Hogg and Elliot Tanis. Probability and Statistical Inference: Fifth Edition. Prentice Hall Inc., Upper Saddle River, NJ, USA, 1997.

[21] Vivek Joshi, Rajeev Rao, Dennis Sylvester, and David Blaauw. Logic SER Reduction through Flipflop Redesign. In Proceedings of ISQED 2006, March 2006.

[22] Srivathsan Krishnamohan and Nihar R. Mahapatra. A Highly-Efficient Technique for Reducing Soft Errors in Static CMOS Circuits. In Proceedings of ICCD, 2004.

[23] S. Krishnaswamy, S.M. Plaza, I.L. Markov, and J.P. Hayes. Enhancing Design Robustness with Reliability-aware Resynthesis and Logic Simulation. In Proceedings of ICCAD 2007, November 2007.

[24] Smita Krishnaswamy, Igor L. Markov, and John P. Hayes. On the Role of Timing Masking in Reliable Logic Circuit Design. In DAC '08: Proceedings of the 45th Annual Conference on Design Automation, 2008.

[25] H. Q. Le, W. J. Starke, J. S. Fields, F. P. O'Connell, D. Q. Nguyen, B. J. Ronchetti, W. M. Sauer, E. M. Schwarz, and M. T. Vaden. IBM POWER6 Microarchitecture. IBM J. Res. Dev., 51(6):639–662, 2007.

[26] M. J. Mack, W. M. Sauer, S. B. Swaney, and B. G. Mealey. IBM POWER6 Reliability. IBM J. Res. Dev., 51(6):763–774, 2007.

[27] D.G. Mavis and P.H. Eaton. Soft error rate mitigation techniques for modern microcircuits. In Proceedings of Reliability Physics Symposium, 2002.

[28] Albert Meixner, Michael E. Bauer, and Daniel J. Sorin. Argus: Low-Cost, Comprehensive Error Detection in Simple Cores. In Proceedings of MICRO-40, December 2007.

[29] Sun Microsystems. OpenSPARC T1 Microarchitecture Specification, August 2006.

[30] Natasa Miskov-Zivanov and Diana Marculescu. MARS-C: Modeling and Reduction of Soft Errors in Combinational Circuits. In DAC '06: Proceedings of the 43rd annual conference on Design automation, 2006.

[31] S. Mitra, M. Zhang, N. Seifert, B. Gill, S. Waqas, and K. S. Kim. Combinational Logic Soft Error Correction. In Proceedings of International Test Conference, November 2006.

[32] Subhasish Mitra, Norbert Seifert, Ming Zhang, Quan Shi, and Kee Sup Kim. Robust System Design with Built-In Soft-Error Resilience. IEEE Computer, 38(2):43–52, 2005.

[33] G.E. Moore. Cramming More Components Onto Integrated Circuits. Proceedings of the IEEE, 86(1):82–85, January 1998.

[34] Shubhendu S. Mukherjee, Michael Kontz, and Steven K. Reinhardt. Detailed Design and Evaluation of Redundant Multithreading Alternatives. In Proceedings of ISCA-29, May 2002.

[35] Shubu Mukherjee, Joel Emer, and Steven Reinhardt. The Soft Error Problem: an Architectural Perspective. In Proceedings of HPCA-11, February 2005.

[36] Chris Myers. Asynchronous Circuit Design. Wiley-Interscience, New York, NY, USA, 2001.

[37] H.T. Nguyen and Y. Yagil. A Systematic Approach to SER Estimation and Solutions. In Proceedings of Reliability Physics Symposium, 2003.

[38] Michael Nicolaidis. Time Redundancy Based Soft-Error Tolerance to Rescue Nanometer Technologies. In VTS '99: Proceedings of the 1999 17TH IEEE VLSI Test Symposium, 1999.

[39] M. Omana, G. Papasso, D. Rossi, and C. Metra. A Model for Transient Fault Propagation in Combinational Logic. In Proceedings of IOLTS, July 2003.

[40] Heidi Pan and Krste Asanović. Heads and Tails: A Variable-length Instruction Format Supporting Parallel Fetch and Decode. In CASES '01: Proceedings of the 2001 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, 2001.

[41] D.B. Papworth. Tuning the Pentium Pro Microarchitecture. IEEE Micro, 16(2):8–15, Apr 1996.

[42] I. Polian, S.M. Reddy, and B. Becker. Scalable Calculation of Logical Masking Effects for Selective Hardening Against Soft Errors. In Proceedings of SELSE, April 2008.

[43] P. Racunas, K. Constantinides, S. Manne, and S.S. Mukherjee. Perturbation-based Fault Screening. In Proceedings of HPCA-13, February 2007.

[44] R. Rajaraman, J. S. Kim, V. Narayanan, Y. Xie, and M. J. Irwin. SEAT-LA: A Soft Error Analysis Tool for Combinational Logic. In Proceedings of the International Conference on VLSI Design, January 2006.

[45] Rajeev Rao, David Blaauw, and Dennis Sylvester. Soft Error Reduction in Combinational Logic Using Gate Resizing and Flip-flop Selection. In Proceedings of the ICCAD, November 2006.

[46] Rajeev Rao, Kaviraj Chopra, David Blaauw, and Dennis Sylvester. Computing the Soft Error Rate of a Combinational Logic Circuit Using Parameterized Descriptors. IEEE Transactions on Very Large Scale Integration Systems (T-VLSI), 26(3):468–479, March 2007.

[47] R.R. Rao, K. Chopra, D. Blaauw, and D. Sylvester. An Efficient Static Algorithm for Computing the Soft Error Rates of Combinational Circuits. In Proceedings of DATE, March 2006.

[48] J. Ray, J.C. Hoe, and B. Falsafi. Dual Use of Superscalar Datapath for Transient-fault Detection and Recovery. In Proceedings of MICRO-34, December 2001.

[49] V. Reddy and E. Rotenberg. Inherent Time Redundancy (ITR): Using Program Repetition for Low-Overhead Fault Tolerance. In Proceedings of DSN, June 2007.

[50] V. Reddy and E. Rotenberg. Coverage of a Microarchitecture-Level Fault Check Regimen in a Superscalar Processor. In Proceedings of DSN, June 2008.

[51] V.K. Reddy, A.S. Al-Zawawi, and E. Rotenberg. Assertion-Based Microarchitecture Design for Improved Fault Tolerance. October 2006.

[52] G.A. Reis, J. Chang, N. Vachharajani, R. Rangan, and D.I. August. SWIFT: Software Implemented Fault Tolerance. In Proceedings of CGO, March 2005.

[53] George A. Reis, Jonathan Chang, Neil Vachharajani, Ram Rangan, David I. August, and Shubhendu S. Mukherjee. Design and Evaluation of Hybrid Fault-Detection Systems. In Proceedings of ISCA-32, 2005.

[54] Eric Rotenberg. AR-SMT: A microarchitectural approach to fault tolerance in microprocessors. In Proceedings of FTCS, 1999.

[55] J.P. Roth. Diagnosis of Automata Failures: A Calculus and A Method. IBM J. Res. Dev., 10:278–291, 1966.

[56] G.P. Saggese, A. Vetteth, Z. Kalbarczyk, and Ravishankar Iyer. Microprocessor Sensitivity to Failures: Control vs. Execution and Combinational vs. Sequential Logic. In Proceedings of DSN, 2005.

[57] Thomas Scherrer. Z80 Family CPU User Manual. http://www.zilog.com/docs/z80/um0080.pdf.

[58] Thomas Scherrer. Z80-Family Official Support Page. http://www.z80.info/.

[59] Norbert Seifert and Nelson Tam. Timing vulnerability factors of sequentials. In IEEE Transactions on Device and Materials Reliability, volume 4, pages 516–522. IEEE Computer Society, 2004.

[60] Sanjit A. Seshia, Wenchao Li, and Subhasish Mitra. Verification-Guided Soft Error Resilience. In Proceedings of Design Automation and Test in Europe (DATE), April 2007.

[61] Premkishore Shivakumar, Michael Kistler, Stephen W. Keckler, Doug Burger, and Lorenzo Alvisi. Modeling the Effect of Technology Trends on the Soft Error Rate of Combinational Logic. In DSN '02: Proceedings of the 2002 International Conference on Dependable Systems and Networks, 2002.

[62] T. J. Siegel, E. Pfeffer, and J. A. Magee. The IBM eServer z990 Microprocessor. IBM J. Res. Dev., 48(3-4):295–309, 2004.

[63] Jared C. Smolens, Brian T. Gold, Babak Falsafi, and James C. Hoe. Reunion: Complexity-Effective Multicore Redundancy. In Proceedings of MICRO-39, 2006.

[64] D. Sorin, M. Martin, M. Hill, and D. Wood. Safetynet: Improving the Availability of Shared Memory Multiprocessors with Global Checkpoint/Recovery. In Proceedings of ISCA-29, 2002.

[65] V. Srinivasan, A.L. Sternberg, A.R. Duncan, W.H. Robinson, B.L. Bhuva, and L.W. Massengill. Single-event Mitigation in Combinational Logic using Targeted Data Path Hardening. IEEE Transactions on Nuclear Science, 52(6):2516–2523, December 2005.

[66] Karthik Sundaramoorthy, Zach Purser, and Eric Rotenberg. Slipstream Processors: Improving both Performance and Fault Tolerance. SIGARCH Computer Architecture News, 28(5):257–268, 2000.

[67] Ivan Sutherland, Bob Sproull, and David Harris. Logical Effort: Designing Fast CMOS Circuits. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[68] A. Taber and E. Normand. Single Event Upset in Avionics. IEEE Transactions on Nuclear Science, 40(2):120–126, April 1993.

[69] Nicholas Wang, Michael Fertig, and Sanjay Patel. Y-branches: When You Come to a Fork in the Road, Take it. In PACT 2003: Proceedings of the 12th International Conference on Parallel Architectures and Compilation Techniques, 2003.

[70] Nicholas J. Wang, Justin Quek, Todd M. Rafacz, and Sanjay J. patel. Characterizing the Effects of Transient Faults on a High-Performance Processor Pipeline. In DSN '04: Proceedings of the 2004 International Conference on Dependable Systems and Networks, 2004.

[71] N.J. Wang and S.J. Patel. Restore: Symptom-Based Soft Error Detection in Microprocessors. IEEE Transactions on Dependable and Secure Computing, 3(3):188–201, July-Sept. 2006.

[72] James W. Watterson and Jill J. Hallenbeck. Modulo 3 Residue Checker: New Results on Performance and Cost. IEEE Transactions on Computers, 37(5):608–612, 1988.

[73] Christopher Weaver, Joel Emer, Shubhendu S. Mukherjee, and Steven K. Reinhardt. Techniques to Reduce the Soft Error Rate of a High-Performance Microprocessor. SIGARCH Computer Architecture News, 32(2):264, 2004.

[74] Neil Weste and David Harris. CMOS VLSI Design: A Circuits and Systems Perspective. Addison-Wesley, Boston, MA, USA, 2005.

[75] Y.C. Yeh. Triple-triple Redundant 777 Primary Flight Computer. In Proceedings of Aerospace Applications Conference, February 1996.

[76] Ming Zhang and Naresh Shanbhag. A Soft Error Rate Analysis (SERA) Methodology. In International Conference on Computer Aided Design, November 2004.

[77] Ming Zhang and Naresh Shanbhag. A CMOS Design Style for Logic Circuit Hardening. In Proceedings of IEEE International Reliability Physics Symposium, pages 223–229, April 2005.

[78] Ming Zhang and Naresh Shanbhag. An Energy-efficient Circuit Technique for Single Event Transient Noise-tolerance. In IEEE International Symposium on Circuits and Systems, May 2005.

[79] Chong Zhao, Yi Zhao, and Sujit Dey. Constraint-aware Robustness Insertion for Optimal Noise-tolerance Enhancement in VLSI Circuits. In DAC '05: Proceedings of the 42nd Annual Conference on Design Automation, 2005.

[80] Quming Zhou and K. Mohanram. Cost-effective Radiation Hardening Technique for Combinational Logic. In ICCAD '04: Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design, 2004.

[81] Quming Zhou and K. Mohanram. Transistor Sizing for Radiation Hardening. In Proceedings of Reliability Physics Symposium, April 2004.

[82] Quming Zhou and K. Mohanram. Gate Sizing to Radiation Harden Combinational Logic. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 25(1):155–166, January 2006.

[83] J. F. Ziegler, H. W. Curtis, H. P. Muhlfeld, C. J. Montrose, and B. Chin. IBM Experiments in Soft Fails in Computer Electronics (1978–1994). IBM J. Res. Dev., 40(1):3–18, 1996.